computer notes

50¢

September '77

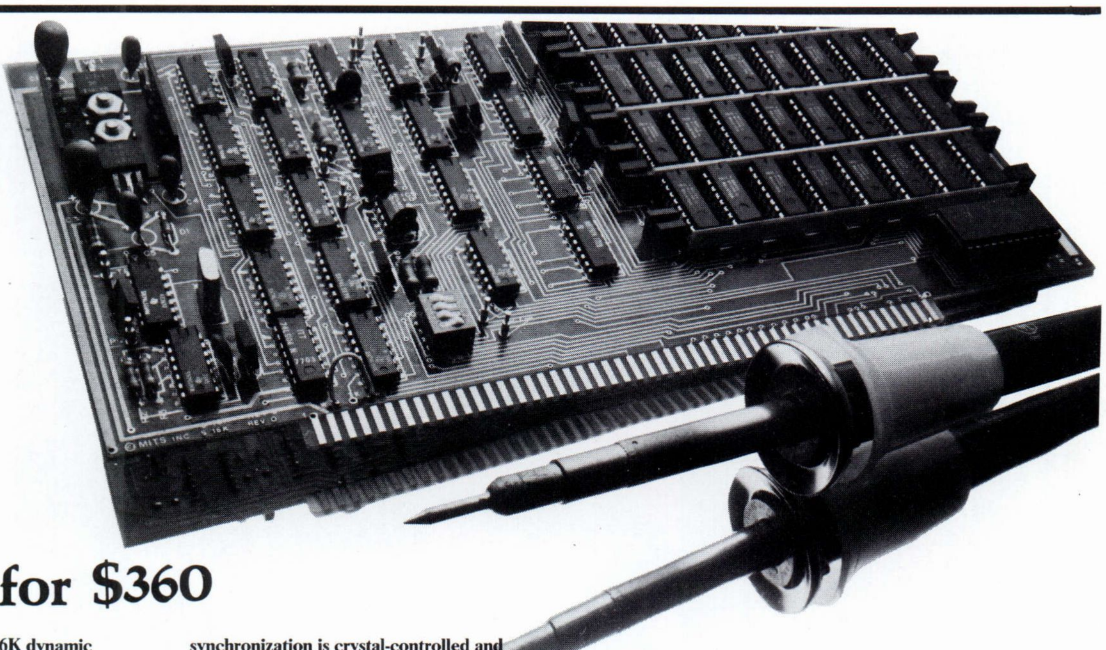
Volume 3 Issue 4

Photograph by **Jim Wiggins**

build your own

CRT . . . Interface . . .

Robot . . . I.C. Test Clip



16K Memory for $360

Unbelievable, but true—a 16K dynamic memory board breaking the $400 barrier. And who would you most expect it from but MITS.

The Altair 88-16MCD offers many outstanding features at a price usually associated with budget products. To begin with, the 88-16MCD can be used in any Altair Bus computer with full compatibility. All refresh circuitry is located on the PC board and receives timing pulses from the CPU. Logic synchronization is crystal-controlled and continuous (no wait states). As with all MITS plug-in boards, the 88-16MCD consumes little power (2.5 watts) and is accessed quickly (RAM access is 350 nanoseconds).

Memory expansion is no longer an expensive proposition when adding the Altair 88-16K Dynamic Memory Board. Build it yourself for $360* or let us do the honors at $395* Either way, it's the best deal in town.

mits

a subsidiary of Pertec Computer Corp.
2450 Alamo S.E.
Albuquerque, New Mexico 87106
(505) 243-7821

*Prices may vary depending on dealer location

SUBMITTAL SPECIFICATIONS

Articles submitted to **Computer Notes** should be typed, double-space, with the author's name, address and the date in the upper left-hand corner of each numbered page. Authors should also include a one-sentence autobiographical statement about their job, professional title, previous electronic and/or computer experience under the article's title. Authors should retain a copy of each article submitted.

All illustrations, diagrams, schematics and other graphic material should be submitted in black ink on smooth white paper. Prints and PMT's are acceptable. No pencil drawings unless properly "fixed." No halftone or wash drawings.

All artwork should be mailed flat, never folded. Unless requested, graphics are not returned. Sketches, roughs and "idea" drawings are generally not used.

Photos, charts, programs and figures should be clearly labelled and referred to by number within the text of the manuscript.

Only clear, glossy black and white photos (no Polaroid pictures) will be accepted. Photos should be taken with uniform lighting and sharp focus.

Program listings should be recorded with the darkest ribbon possible on blank white paper. A paper tape for each program submitted **must** also be included.

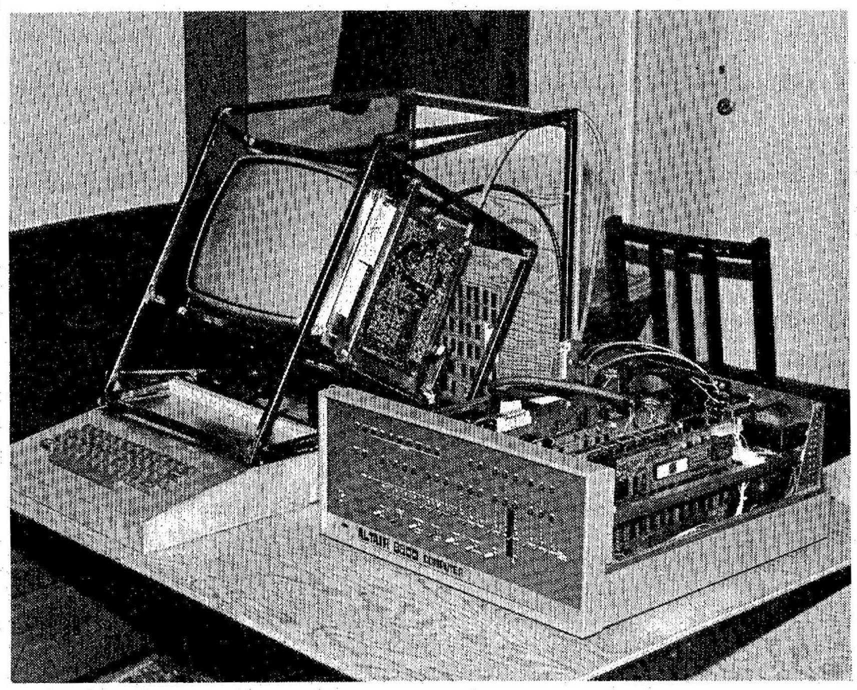
COMPUTER NOTES is published monthly by **MITS, Inc.,** 2450 Alamo SE, Albuquerque, NM, 87106, **(505) 243-7821.** A free year's subscription is included with every purchase of an Altair™ computer. Regular subscriptions can be ordered from the **MITS** Customer Service Dept. for **$5** per year in the U.S. and **$20** per year for overseas. Single copies are available for **50¢** each at all **Altair Computer Centers.** Entire contents copyright, 1977, **MITS, Inc.** Send articles, questions, comments and suggestions to **Editor, COMPUTER NOTES, MITS, Inc.**

MITS, Inc. 1977 (Volume 3, Issue 4, September, 1977)
a subsidiary of **Pertec Computer Corporation**
2450 Alamo S.E., Albuquerque, New Mexico 87106

NEED AN INEXPENSIVE CRT? Build One Using an Altair 8800-Compatible Interface Card

By Jim Wiggins
400 Pemberton Terrace
#116
Kamloops, B.C. Canada V2C 1T3

Wiggins is an avid microcomputer hobbyist who plans to return to school to make computers his vocation.



The CRT terminal and the floppy disc system currently seem to be the most desirable peripherals for microcomputer users. However, they are also the most expensive and for this reason the literature is replete with techniques for hobbyists to inexpensively built or buy their own. The following article describes how a relatively inexpensive, high-quality CRT terminal can be constructed using a commercial Video Ram (VRAM) and an Altair 8800 compatible interface card.

The heart of the terminal is the VRAM ($390), which is manufactured by:
Matrox Electronic Systems
P.O. Box 56, Ahuntsic Stn.
Montreal, Quebec
Canada, H3L 3N5

The card itself, which unfortunately will not fit inside the Altair Computer case, consists of 2K of memory for a display of 24 lines of 80 characters per line with each character being displayed in a 7 x 9 dot matrix format. The character set used in the standard MTX 2480A is the 128 character ASCII+Greek. Custom designed as well as several other standard character fonts are also available. Because the required video signal is generated from the on-card memory, DMA is not required, and the card is read from and written into as if it were normal processor memory. Although the card contains 2K of memory, it appears as 4K on the address bus. This is due to the address decoding scheme. (See Fig. 1). Address lines A6 to A0 inclusive access columns 0 to 79, and address lines A7 to A11 access lines 0 to 23, while column addresses 80 to 127 and line addresses 24 to 31 are ignored. In a sense, this wastes memory address space,

continued

EDITOR
Andrea Lewis

ASSISTANT EDITOR
Linda Blocki

PRODUCTION
Al McCahon
Steve Wedeen
Beverly Gallegos
Lucy Ginley

CONTRIBUTORS
Dave Antreasian
Bruce Fowler
James A. Gupton, Jr.
Robert Rossum
Mike Smith
Jim Wiggins

IN THIS ISSUE

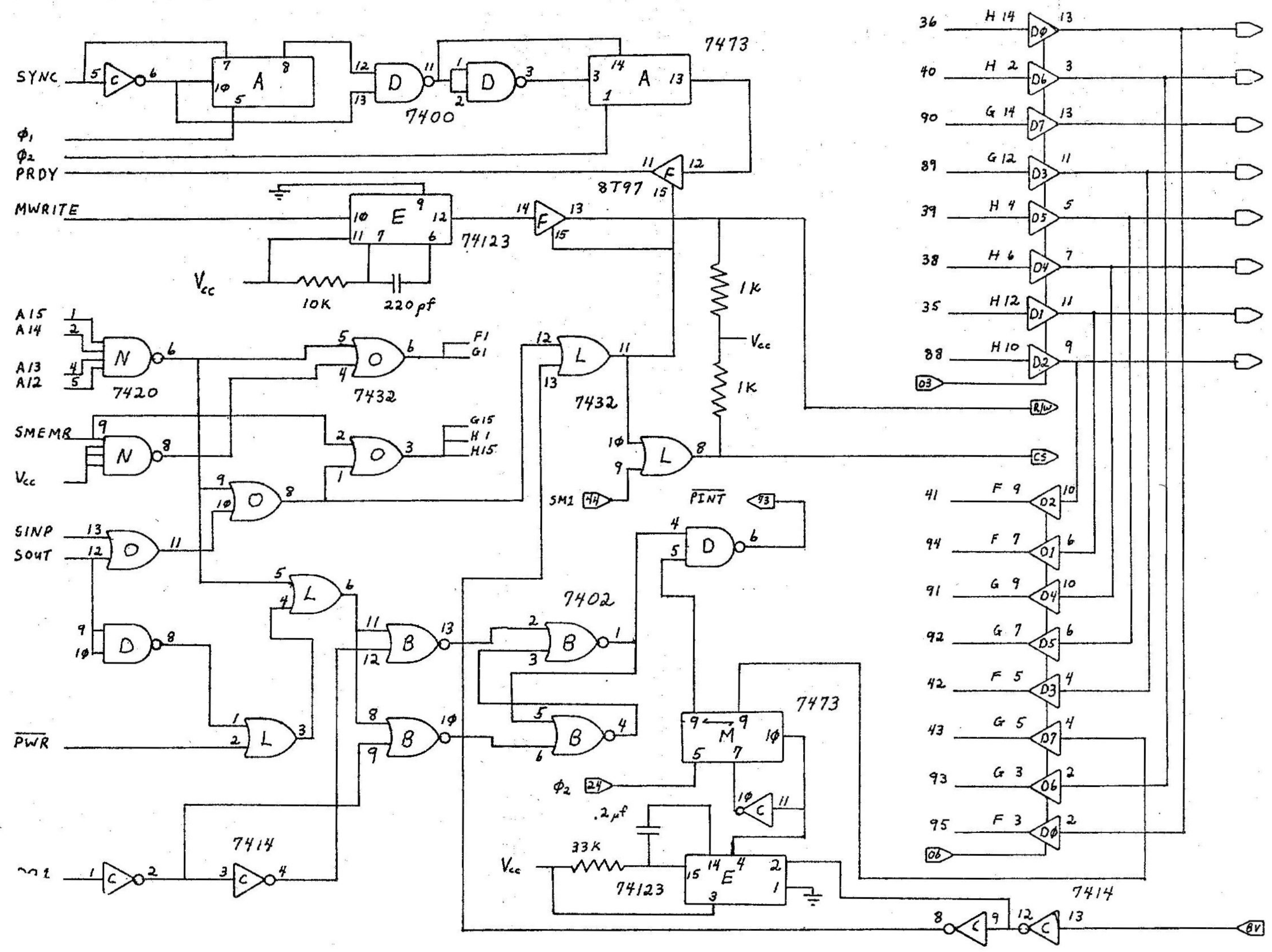
but it isn't a real concern unless you have 62K memory.

Another important feature of the card is the data bus, which is bidirectional and 9 bits wide. Bits 0 through 6 comprise the character code, and bits 7 and 8 are used to select one of four display modes (for each character): normal (00), half intensity (10), inverse video (01), and blink at 1 Hz rate (11). In order to simplify the interface card, I have tied bits 7 and 8 together so that characters are either normal or blinking. However, other combinations, such as normal/inverse or inverse/blink, can be obtained quite easily.

In addition to the address and data lines, there are several other input and output lines from the VRAM used by the display monitor and the interface card. Since both the composite (logic plus sync) and the logical video signals as well as the horizontal and vertical blanking pulses are available, there are only a few restrictions on the type of monitor used. One restriction is that the monitor must have a bandwidth of 10 Mhz minimum. It must also have a long persistance phosphor if the 128 character set fonts are used. The one 64 character font (5 x 7 dot matrix), upper case ASCII, is the exception to this in that a monitor with a standard phophor may be used. For readability, Matrox recommends that the monitor be 12 inches or larger. The vertical blanking (BV) from the VRAM is also used by the interface card as a status signal to control access to the VRAM card itself.

Although the VRAM may be read from or written into during the beam trace time, a noticeable flicker results. For this reason, the BV signal and the interface card ensure that access to the card is during vertical retrace (4.61 msec). The chip select ($\overline{CS}$), which is active low for read and write, and read/write (R/W), which is active low for write, are the only control signals from the interface card to the VRAM required for operation. Although the actual timing relationships are not shown here, they are similar to those for the memory chips themselves, i.e. 2102-1. The minimum read and write cycle times for these chips are 500 ns, which makes it technically possible to access VRAM without memory wait states. However, because the VRAM is separated from the interface card by six feet of ribbon cable, two wait states are used to ensure that data is stable during the write pulse or the processor read pulse DBIN.

The interface card itself is shown in Figure 2 with IC A used to request the required wait states. Before examining the circuit, I would like to thank MITS for their indirect assistance in the design of this interface card. Owners of the MITS 1K Static and S10B cards will note a distinct resemblance in the decoding, wait state generation, and interrupt enable circuitry.



(Apart from these areas of similarity, the design is all mine; patents are pending.) Not shown on the schematic of Figure 2 are 2 hex buffers for the 12 address lines and pull-up resistors on the 8 data lines. While the Altair computer uses separate input and output busses, the VRAM uses a bidirectional bus which is split, using 8T97s, ICs, F, G, and H. Four gates of F and 4 of G comprise the data in, and 6 of H and 2 of G are used for data out. To the left of the schematic, the ICs N and 0 select either the data out or data in lines, depending on the states of SOUT, SIMP, SMEMR and, of course, the address lines A12 to A15 (IC N6). DI is selected through IC 0 pin 6, and DO is selected through IC 0 pin 3. IC 0 pin 8 is active low during both read and write operations. It is this signal applied to IC L pin 12 plus the BV signal applied to IC L pin 13 that produces the chip select ($\overline{CS}$) pulse at IC L pin 8. $\overline{CS}$ is also, possibly unnecessarily, a function of M1, because IC 0 8 will go low during the latter part of M1. IC L pin 11 also enables two gates of IC F via pin 15 of F. The R/W signal is applied through one to the VRAM, and the other is the source of the PRDY signal to the processor.

There's also output port on the interface card. It's used for interrupt enable, which consists of the RS flip-flop comprised of the 4 gates of IC B. Data on the D01 line sets or resets this FF, and the effective write pulse for the port is generated when IC L pin 6 goes low. The output at IC L pin 6 is a function of $\overline{PWR}$ (IC L 2), SOUT (IC D 9), and the port address applied to pins, 1, 2, 4, and 5 of IC N. In this situation, it is the 4 most significant bits of the port address (which appears on both the upper and lower halves of the address bus) that cause the port to be selected. IC N 6 thus performs the dual purpose of selecting the output port as well as the memory space of the VRAM. This scheme does restrict users to 240D output ports, but that should be more than sufficient. The execution of the instruction OUT 350 will, therefore, set or reset the FF, the output of which is applied to enable the interrupt gate at pin 4 of IC D. The signal that produces the interrupt is a function of the BV and is generated by IC E and M. The period of retrace is 4.61 msec, which is shortened to about 3.5 msec by IC E and is synchronized with phase 2 by IC M. The output of IC M at pin 9, which is active high, thus generated the interrupt at IC D 6 during retrace. This signal is also fed to IC G 4, DI7, where it can be monitored by such memory reference instructions as MOV, A and M.

The pulse at IC M 9 is shorter than BV, so D17 can be sampled during the final microseconds of that pulse. In that case, when the card is accessed, even though IC M 9 is low, IC L 13 will still be active low, and the $\overline{CS}$ pulse can still occur.

The following program briefly illustrates the method of operation. The program simply fills the screen with a given character and loops.
00 MVI

```
00 MVI        076    Set accum with desired character
01 DATA       067    ASCII 7
02 STA        062    Store accum
03 L          100
04 H          003
05 LXI        001    Set reg pair B with line increment
06 C          200    this is one in bit 7
07 B          000
10 LXI        041    Set reg pair H with starting address of display
11 L          000
12 H          360
13 MOV A,M    176    Move data byte form interface card to accum
14 ANI        346    Clear off everything but bit 7-retrace status
15 DATA       200
16 JZ         312    If zero not in retrace, so loop till retrace
17 L          013
20 H          001
21 LDA        072    As in retrace, load accum with data
22 L          100
23 H          003
24 MOV M,A    167    Move accum to VRAM
25 INX        043    Increment column/line counter in reg pair H
26 MOV A,L    175    Move reg L to accum-contains column count
27 ANI        346    Clear off bit 7, which is part of line count
30 DATA       177
31 CPI        376    Compare column count with 80D-last plus one of column
32 DATA       120
33 JNZ        302    If not zero, then more on line to go
34 L          013
35 H          001
36 MOV A,L    175    Line completed, so column count in L to accum
37 ANI        346    Clear off all but bit 7, which is part of line count
40 DATA       200
41 MOV L,A    157    Move cleared column count back to L
42 DAD        011    Increment line count by double add with reg pair B
43 MOV A,H    174    Move line part of address in H to accum
44 CPI        376    Compare with last line plus one
45 DATA       374    This is 360Q plus 140, which corresponds to line 24D
46 JNZ        302    If not zero, then more lines to go
47 L          013
50 H          001
51 JMP        303    Screen filled so loop
52 L          000
53 H          001
```

This combination of Matrox 2480 with the Altair 8800 compatible interface card has resulted in an excellent but relatively inexpensive CRT terminal. Although I used a separate interface card (partially because of impatience), other users may not have to, since Matrox has been working on a card compatible with Altair computers and may now be producing it.

The benefit of a separate interface card (apart from space considerations) is the freedom in being able to choose the aspects of the interface which are the most important, as I have done with the interrupts and bits 7 and 8. Requirements will differ among people, and some may, for example, decide to access VRAM during horizontal retrace, which would about treble the data transfer rate. Using vertical retrace, the data rate is about 6000 characters per second; using horizontal retrace, rates of 18000 characters per second and greater could be achieved. As another example, latches could be implemented to use the full 9-bit data bus for reading and writing. The interface card described here, while more than adequate for my purposes, is therefore only one of many possible ways of building a viable CRT terminal with the Matrox VRAM.

CAREFUL BULL IN THE CHINA SHOP
A Cheap Approach to the Mechanics of Robotics

This is the first article in a three-part series on building a robot. Part II will cover in detail the mechanics of robot building, and Part III will discuss applications.

Members of the United States Robotics Society (USRS) are using the family name "Rossum" as a kind of collective pseudonym for their publications. Members who prefer to be anonymous may publish through USRS under whatever "Rossum-name" they reserve. Thus far, half a dozen names have been spoken for, e.g. "S.A. Rossum," "D.I. Rossum," and some folks whose real family name is Rossum have been listed.

"Robert Rossum" is a writer of books, articles, and non-theatrical motion pictures, who has spent most of the past 20 years working in research and developmental laboratories.

Part I

By Robert Rossum

Someone once said that the most interesting thing computers ever do is to blow hot air on your shoes while they hum and soak up money. An intelligent machine, no matter how clever, lacks charm if it just sits around like a bump on a log. Perhaps part of the current enthusiasm for robotics is a reaction to this static performance of our clever machines. Roboticists almost universally report their determination to construct mobile systems.

The ordinary roboticist is usually a good thinker-upper, programmer, planner, and innovator but seldom a first-rate mechanical engineer and master machinist. Although drawing conceptual plans for experimental mechanical systems is a necessary first step, actual construction and modification of mechanical creatures is prohibitively expensive in cash and time. The mobile systems built by institutions and private workers tend to be awkward, fragile, unstable, and uninteresting as well as expensive. The interesting machines that receive national publicity tend to be anthropomorphic monsters. One such recently publicized system is over six feet high and weighs several hundred pounds. It performs some remarkable tricks under the remote control of its master but looks mighty unstable on its small base. Watching it causes the uneasy feeling that if it dropped a wheel off the edge of a walkway, it would topple over, crushing dog, child, mailman, or Volkswagen. The publicity arising from that incident might not bring cheer to other roboticists.

Even the cute little wheeled systems that experimenters set to snuffling around their laboratories have no more athletic prowness than is required to climb over a doorsill or up on a rug without stalling or upsetting. Conventional mechanical systems are generally proving unsatisfactory for devices that are intended to simulate the performance of living things.

The flaw in the simulation is not chiefly the lack of intelligence. David Heiserman, author of BUILD YOUR OWN WORKING ROBOT, has observed that his robots acquire behavioral characteristics of living creatures, responding to their environment in a surprisingly complex fashion. In his book, Heiserman said the fact that impresses him most is the simplicity of the circuitry involved. He said a few basic sensory channels, simple reflexes, and a trifle of logic allow his machines to behave like simple animals. It may be that the devices are intellectually trivial, but since they can move, displaying their characteristics overtly, and can alter their performance in response to a changing environment, they are interesting. Heiserman's mechanical systems are quite crude, but at least they do something.

If experimenters today can develop cheap and dirty mechanical systems that any clumsy amateur can build in his own garage, the progress in robotics will be significant in the next few years. The purpose of this series is to briefly describe a cheap, not inexpensive, but **cheap** mechanism for many robotics applications. No detailed designs are offered, but roboticists will be able to use the basic principles of the system without further elaboration here in print.

In this electronic age, we think of robotics mechanisms in terms of electronically-controlled servosystems, stepping motors, and complicated, heavy gear trains. Consider servos. Since no mechanical system if perfectly accurate, we must always provide a trial-and-error system that will let a free-moving device accomplish its tasks in spite of imperfection. For example, if you set your pet robot on a course for a fire hydrant a block away, you can be sure that the critter will miss the fireplug unless it knows one when it sees one and can hunt around as necessary to find the thing. Just aiming straight from where you are to the hydrant won't work, since irregularities in the pavement, uneven wear in the robot's wheels and gears, bad aim, or a dozen other problems will almost inevitably prevent the machine from going directly from one place to the other. The robot must be able to correct its course, to "zero in" on the target. Of course, when the machine changes its course, the correction is not likely to be prefect. It may over-correct or encounter more problems along the way. Ordinarily, a servomechanism is employed to make up for imperfections in the rest of a machine, to make the back-and-forth corrections necessary to guide or position a machine properly. The servo is precise in that it takes the machine to exactly the right position. But it's not necessarily "accurate," since it doesn't follow a detailed set of instructions to get to a target.

The distinction between precision and accuracy is important. If your robot is accurate, you may give it instructions such as: Move exactly north 315 feet and five inches. Then make a 90° turn to the left (not an 89 turn or a 91 turn but a 90 turn), and move exactly 19 feet and seven inches. Stop there, or you'll smash your little lens on the knobby thing that is sticking out of the hydrant.

What are the chances that you really know exactly what the instructions should be and that your robot can carry them well out enough to get within six inches of the hydrant? Not very good, unless you have an uncommonly well-made, expensive machine (equipped with a magnificent inertial guidance system, perhaps) working in an environment without obstructions. If there are cracks in the sidewalk, you're in trouble.

If your robot is equipped with sensors and servos, it can use instructions more like this: Move along the sidewalk to the north without falling off the edge or bumping signposts until you detect something that looks like a fireplug off to the left, about 300 feet along the way. Then move toward that hydrant until you're six inches from it. Stop there.

Chances are good that the robot will go precisely where you want it to go. That's precision, not accuracy. The robot may be constructed with only lousy components, may not be able to run accurately within five degrees, may be off by three percent in its judgment of distance, but it will do what you want it to do. Remember that living things are built entirely of lousy, individually unreliable, and irregular components. Even the brain is constructed of stuff that couldn't meet military specifications for purchasing, regardless of actual performance.

Remember, too, that when an animal lifts its foot, it does not usually have to swing that foot clear around a 360 arc to return it to its starting position. Feet move forward and back, up, and down. Tails move to and fro. Muscles in living creatures are paired. Your bicep pulls your forearm up, and your tricep pulls it back down. Mechanical servomechanisms usually work with paried motors, pulling things first one way, then the other, "zeroing-in." The builder is usually depressed by the realization that almost everything in his critter must be duplicated --all motors matched or at least reversible. One common ploy is to make the motor pull against a spring that returns a limb to "normal" position after the motor moves it.

Robot designers usually provide a motor for an arm, a motor for a head, a motor for wagging the tail, etc. or very complex, heavy, power-consuming gear systems to accomplish all these functions with a single motor. But let's consider an alternative -- the ancient double windlass mechanism. Its virtues for the roboticist are many. (See any encyclopedia; look under "captain.")

BASIC SYSTEM

Figure 1 (the motor)

This box with an "M" on it is a motor.

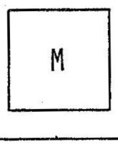


Figure 2 (motor with shaft)

A long shaft protrudes from the motor.

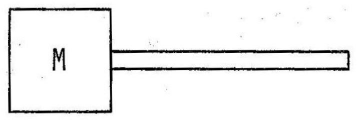


Figure 3 (pulleys added to shaft)

We may place a pair of pulleys on the shaft.

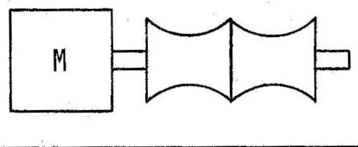


Figure 4 (upper lever added)

Above the shaft at some arbitrary distance is Lever A, pivoted at its center.

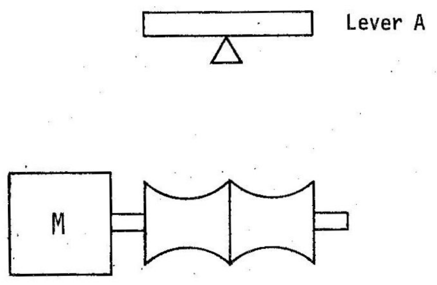


Figure 5 (lower lever added)

Below the shaft is Lever B, also pivoted at its midpoint. Our interest here is in getting Lever B to do something in particular when we move Lever A.

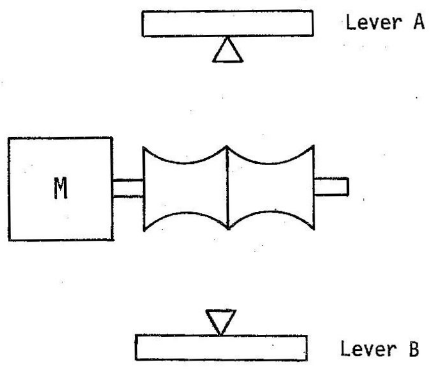


Figure 6 (Cords C1 and C2 added)

We connect Levers A and B with Cords C1 and C2. The cords are wrapped loosely around the pulleys on the shaft so that when the motor turns the pulleys just spin inside the loose cords without affecting them and the levers.

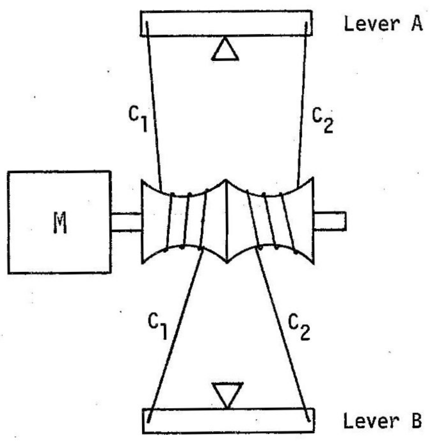
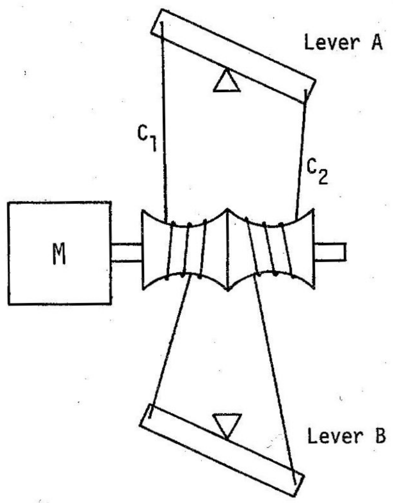


Figure 7 (both levers canted to same angle)



Suppose that you take hold of Lever A, tilting it upward at the left end. That pulls Cord C1 tight around its pulley, but Cord C2 remains loose around its pulley. Here the mechanical magic begins. As Cord C1 grips the pulley, the force of the motor begins to pull on the cord. Even if you lift the end of the lever very delicately with your fingertips, the cord, hence also the end of Lever B, will be pulled by the full force of the motor. You need only keep a bit of tension on the top part of that cord to apply the motor's full force to the task of lifting up the end of Lever B.

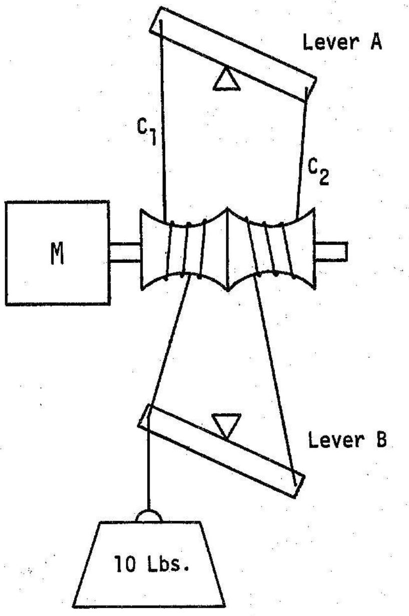
If you pull the end of Lever A steadily up to some particular position, the motor will wind up the lower part of Cord C1 until Lever B is cocked at the same angle as Lever A. Then the cord will begin to slip on

continued

the pulley, and the pulling force of the motor will be relieved. You have applied a small control force to the upper lever, causing the motor's force to be applied to the lower lever. In fact, a weight of some significance might be hanging from that left end of Lever B.

Figure 8 (10 lb. weight hanging from Lever B)

This is far more weight than you could lift with your fingertip. The motor would do the lifting, multiplying the control force greatly.



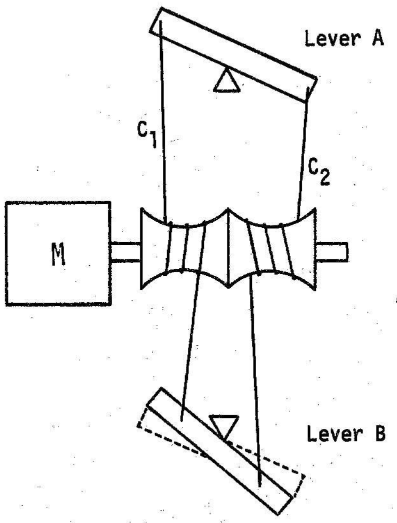
Notice that when the Cord C1 begins to slip, C2 is just on the point of growing tight. When the action stops, the windings of the two pulleys are slightly loose, just as they were when the action began. The system is all ready to perform again promptly when another control force is applied to a lever. If you pull up on the right end of Lever A now, Lever B will be returned to its original matching position -- sort of a bicep-tricep action.

You've done two things -- controlled the position of Lever B by manipulating Lever A and multiplied the tiny control force with the force of the motor. These are both very important to the roboticist who is hoping to control the limbs of a mechanical creature.

SOME MORE BASICS

You may choose to amplify your **motion** as well as your control force.

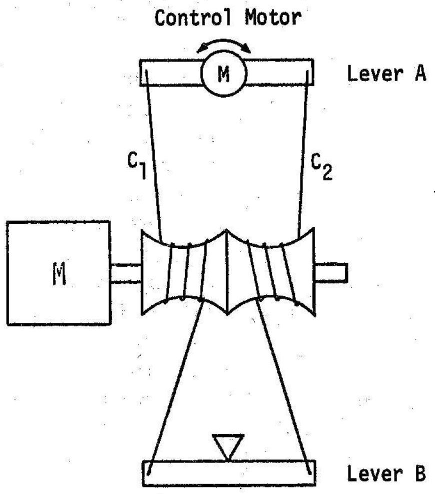
Figure 9 (Cords on Lever B now attached appreciably closer to the pivot than on Lever A)



If you now raise the left end of Lever A the same distance you did before, the force of the motor will be applied to Lever B in the same way. The same length of cord will be drawn up by the pulley, but the left end of Lever B will be moved a greater distance. You have multiplied both force and motion.

The motor here may be as large as you like, depending upon the application. The control force you apply to Lever A may, in fact, be supplied by another motor, since your robot will probably employ an electrical system, and turning power off and on in electrical motors will be a straightforward matter. The control motor may be small, both in physical size and power. (The main motor may even be gasoline or steam powered, if you like, depending on your application and willingness for your robot to breathe real smoke and fire with a variety of associated noises.

Figure 10 (control motor on Lever A)



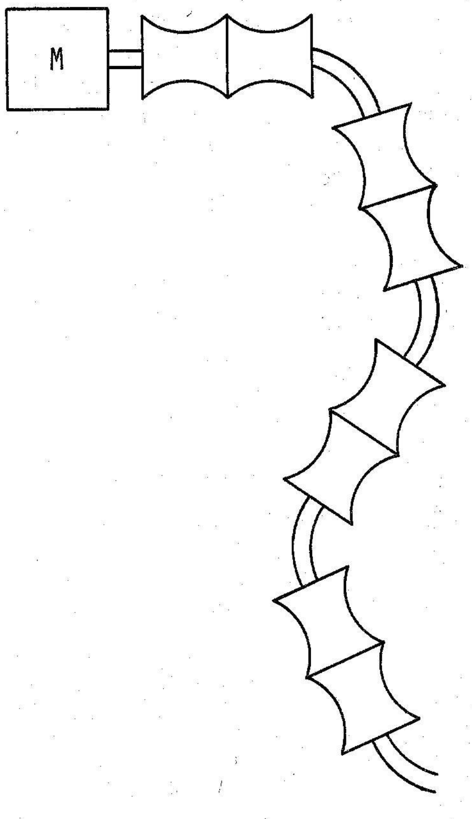
In fact, your control motor might sensibly by a reversible shaded-pole motor. Motor experts say that a shaded-pole motor can be held in a stalled condition indefinitely without damage, and that's an advantage. (As later article will discuss a mixed bag of alternative to control motors.) With signals from your robot's brain, presumably your personal computer, you can move Lever B either way automatically with appreciable force.

Figure 11 (main motor shaft with more sets of pulleys)

The shaft from the main motor may be equipped with numerous pairs of pulleys so that power may be applied at any point along the shaft to any chosen lever down below.

Figure 12 (flexible shaft with pulleys along its snaking path)

The shaft may be flexible so that power can be transmitted from the main motor to remote regions of the robot in which it resides.



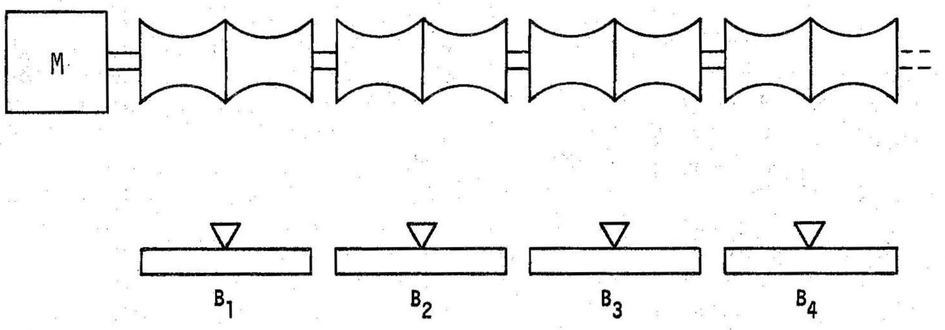
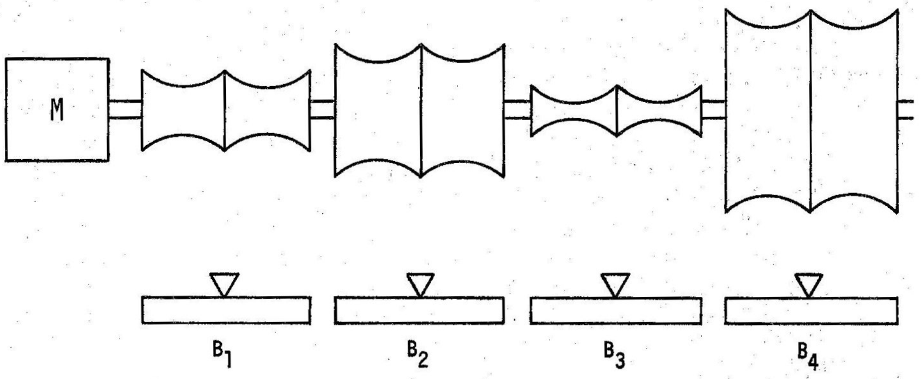


Figure 13 (various sizes of pulleys on the shaft)

All the pulleys may be of different sizes on this same shaft so that Lever B1 may be moved with a different amount of power from that applied to B2 and so on. Maybe you don't want the robot to wag its tail with enough force and speed to smash a chair leg. You can control the speed and power of the way by choosing levers of appropriate length and pulleys of appropriate diameter.



An important consideration at this point is shared power. Obviously, there's a limit to the number of pulleys you can put on the shaft of a given motor. There's a practical physical constraint of some kind to balance your every wish. If you tighten the cords at every point along the shaft, drawing power from the motor at each pair of pulleys, your chances of overloading the motor are very great. But there's the beauty of the system (well, one beauty among many) -- it works like an animal. Like any animal, you rarely use all of your muscles at once. When you run, you may be using your leg muscles in an extreme fashion, but you are not simultaneously using your neck and arm muscles to their fullest extent. Chances are that you are not also trying to bite through a heavy bone, too, drawing a great deal of energy in your jaw muscles.

That's a very significant factor in the design of animals. You have a certain amount of chemical energy stored locally in your muscles. When you move the muscles, you consume some of that available energy. If you exert the muscles greatly, you use up all that's available locally, and must get more sugar from the liver. With great exertion you can develop a severe shortage of energy locally. Luckily, you seldom exert all muscles at the same time, so you don't develop a general deficit of energy. (However, people do sometimes die of overexertion. That's one of the problems for people stuck in blizzards. They tend to use up all of their reserves struggling through the snow, and then lie down to rest. When they quit moving, they quit pumping new chemicals to their depleted muscles fast enough. The cold and lack of energy may be fatal.) The analogy is not perfect, but it's pretty good. This double windlass system allows the energy of the main motor to be shared by many functions in the body of the robot. The **average** load on the motor can be quite low, while large amounts of energy are rapidly available wherever needed. When separate motors are used at all places where energy is needed, those motors must be big enough to supply all the energy that will **ever** be needed from them. That means a lot of extra weight is being dragged around all the time just in case a burst of energy is needed. The double windlass system solves much of this problem with a comparatively simple simulation of the system Nature has been using effectively for a long time.

I'm not complaining about standard mechanical systems. There's much to be said for the clever designs that competent engineers have developed for robotic and non-robotic mobile systems using modern technology. However, cheapness is not a feature of standard mechanical systems, and the average home craftsman simply can't cope with them.

The double windlass system can be assembled by the home experimenter with Tinkertoys or an Erector Set. The interested roboticist can work with this system himself even before the next article in this series on more mechanics of robot-building is published next month.

The pulleys can be empty thread spools in the experimental system. When you get around to building a rig that's meant to last, you'll want to use metal, because there's a lot of wear. Don't the cords stretch? Sure, and they'll have to be tightened once in a while. So what? At least you can figure out what's wrong and fix it yourself. (And there will be many maddening problems inherent in this system as in any other.) By machinists standards, the whole mechanism can be quite sloppy and still work. Precision can be achieved in a sloppy system without accuracy.

In the discussions leading up to this article someone asked: "Isn't there a real safety factor in the fact that the cords will slip on the pulleys if they are overloaded?"

"Oh, no. The cords will break before they slip. This is the kind of mechanism people use to pull two or three miles of oil drill stem up out of wells. The windlass is a powerful tool. Why do you ask?"

"Well, I guess I don't want the robot to be too strong."

continued

BUILD YOUR OWN INTERFACE
Tips from a professional

By Mike Smith

Smith is a professional design engineer and avid computer hobbyist. His article deals with Altair bus signals and three different types of I/O: Flag Testing, Program Interrupt, and Direct Memory Access. This story first appeared in the June 1977 issue of KILOBAUD. Copyright 1977 KILOBAUD Publications, Inc., Peterborough, NH, USA. All rights reserved. Reprinted by permission.

One of the most interesting and rewarding areas of the personal computer hobby is that of designing and building custom interfaces. Aside from the sheer pleasure and satisfaction that comes with seeing your new design work, there is the possibility of interfacing something that has never been interfaced as well as the benefit of substantial money savings as compared to the purchase of a commercial kit or finished product.

This article deals specifically with interfacing to the Altair 8800 series of computers and applies equally to all three models (8800, 8800A, and 8800B). The first step toward successfully interfacing any computer is a thorough understanding of the bus. The Altair Bus is a 100 line printed circuit board bus, in which all like numbered connector pins connect to one another via etched copper lines. This structure allows any interface, CPU or memory printed circuit board to be inserted into any vacant connector slot. The 8800A and 8800B come equipped with an 18-slot bus (mother board), while the 8800 is provided with a 4-slot bus with space provided for 3 additional 4-slot mother boards.

Each of the 100 lines on the Altair Bus has a predefined function which must be fully understood in order to make good use of the bus.

Table 1 contains a complete breakdown of all the Altair Bus signals, given in functional logic notation. This means that in each signal mnemonic there are two parts. The first is the signal abbreviation and the second is the active level representation. The two parts are separated by a hyphen for clarity. The active level representation is in the form of an upper case H for active high and an upper case L for active low. Functional representation does not apply to power and ground lines. To correlate the functional logic shown here to the "positive logic" symbology shown in the Altair documentation, drop the active level representation and draw a Boolean NOT sign over the tops of these signals shown here as L. The functional notation is a far better approach because it reserves the use of the Boolean NOT symbol for use when NOT is intended. Positive logic notation on the other hand, uses the NOT symbol every time an active low is indicated, so that every time the logical NOT function is desired, it will be confused with active low.

Let's take a look at the major groups of signals on the bus. The **address** lines are outputs from the CPU board and are inputs on all memory and I/O boards. The I/O boards use only the lower 8 bits (A0-H through A7-H) because during I/O transfers, the upper 8 bits are identical to the lower 8 bits. In the event that direct memory access (DMA) is used, the DMA controller must also generate an address on the address lines. In the 8800 and 8800A, these lines (through current limiting resistors) provide the drive for the Address indicators on the front panel. This seriously limits the high going drive of the 8T97 drivers on the CPU board and therefore it is suggested that each board receive the address lines using only one low power TTL or low power Schottky TTL device. In the event that a DMA controller is being designed, it is recommended that address drivers equivalent to the 8T97s on the CPU board be used.

The **data-out** lines are outputs from the CPU board and are inputs on all memory and I/O boards. In the event that DMA is used, the DMA controller must also drive the data-out lines when the direction of the data transfer is to the memory from the DMA controller. Assuming no more than the 18 cards that will fit in the cabinet are connected to the bus, up to 2 milliamps of low loading may occur on each board in the bus. This equates to one standard or Schottky load, 5 low power Schottky loads, or 10 low power TTL loads. A DMA controller should drive these lines with 8T97 Tri-state buffers or equivalent.

The **data-in** lines are inputs to the CPU board and are driven from the memories and I/O boards. In the event that DMA is used, the DMA controller is an additional input for the data-in lines. Also, the front panel is an input for these lines. Since the front panel and CPU both use 74LS04 receivers for the data-in lines and the lines are "pulled-up" using 1k Ohm resistors at the CPU board, almost any Tri-state or open collector TTL driver may be used to drive the data-in lines from the memory and I/O boards. However, to insure optimum noise immunity and capacitive drive over the length of bus, it is recommended that the 8T97 type buffers be used to drive the data-in lines.

The **status** lines are sent out to the bus from the CPU board (and from the DMA controller is installed). The status lines consist of 8 lines which are selectively used by the memories and I/O boards to obtain information about the nature of the cycle. Those lines are also displayed on the front panel. The status lines are electrically of the same nature as the address lines, which means that they should be loaded by only one low power TTL or one low power

CAREFUL BULL IN THE CHINA SHOP

continued

"Too strong for what?"

"For people. I don't want it to hurt anybody by accident, and I though maybe the cords would slip in case the machine happened to be gripping somebody too hard."

"Ah. Well, you'll have to take care of that in the machine's logic. I suppose you could build in a sensing circuit that makes it turn off when it hears a scream."

"That isn't the comfort I was looking for."

"Sorry. A machine is a machine. Build it the way you want to build it. Maybe it can learn to be careful."

I haven't discussed the logic, brain, or reflexes of a robot in this article, though some of those matters will be touched upon later in the series. Instead, I've offered a cheap and dirty approach to making robots **do** something interesting. If you have been stewing in frustration over your inability -- financial or mental -- to build a working system to go with the brains on your shelf, get busy with the spools and Erector Set motors.

Schottky TTL load per board. A DMA controller must be capable of generating these signals during a DMA transfer. These lines are SINTA-H, SWO-L, SSTACK-H, SHLTA-H, SOUT-H, SM1-H, SINP-H, and SMEMR-H.

The **processor** lines are a buffered group of inputs and outputs which are derivatives of the Intel 8080 processor signals. The 6 processor output lines are PSYNC-H, PDBIN-H, PWAIT-H, PWR-L, PHLDA-H, and PINTE-H. The DMA controller has control of these lines during a DMA transfer. The 5 input lines are PRDY-H, XRDY-H, PHOLD-L, PINT-L, and PRESET-L. Since some of the output signals in this group are used to drive front panel indicators, the input loading should not exceed one low power TTL or one low power Schottky TTL load per board. The input signals are all "pulled-up" using 1k Ohm resistors so that they may be driven by any Tri-state or open collector TTL gate or buffer. Although many kits (and the Altair manual itself) advocate the use of the PRDY-H line for inducing "wait" states, a much better and electrically correct) way of doing this is to use the XRDY-H line. The front panel drives PRDY-H with a constantly enabled 8T97 which has substantial haigh going drive capability. To pull this line down while the front panel is pulling it high causes large instantaneous surge currents in the devices, causing unnecessary noise spikes as well as abuse to the devices themselves.

The **disable** group of lines is used to disable the various Tri-state drivers on the CPU board when a DMA cycle occurs. The DMA controller then becomes responsible for driving the disabled lines. There are 4 disable signals which are used with DMA. They are STADSB-L, C/CDSB-L, ADDDSB -L, and DODSB-L. A fifth disable signal is SSWDSB-L, which has nothing to do with DMA, but instead is used to gate in the sense switches when an IN 377 (octal) instruction occurs.

The heart of the Altair 8800 system is the CPU board. A block diagram of this board is shown in Fig. 1. This block diagram is provided to give an overview of the CPU connections to the bus. All references of **in** and **out** in the Altair system are with respect to the CPU board. The bidirectional data lines of the 8080 microprocessor are split into data-in and data-out lines for use on the Altair bus. The contents of the bidirectional data lines are latched into the 8212 latch at SYNC time by the O1 signal. The outputs of the 8212 latch are buffered for system use by 8T97 Tri-state buffers. The processor output signals are also buffered using 8T97s and

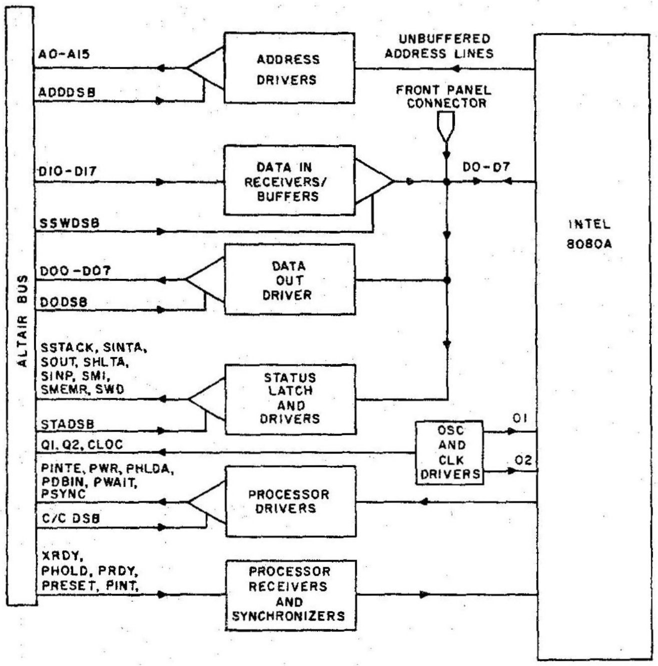


Fig. 1. CPU Block Diagram.

presented to the bus. The processor input signals are also buffered using 8T97s and presented to the bus. The processor input signals are passed through receivers and then presented to the 8080 chip. The ready signals (PRDY-H and XRDY-H) are ANDed, synchronized with Ø2, sent to the 8080. PHOLD-L is also synchronized with O2 before being sent to the 8080. The CPU block diagram should also be of value in the event that troubleshooting the CPU becomes necessary.

Fig. 2 is a block diagram of a typical 4K byte static memory as implemented for the Altair bus. In the static memory board, the control is relatively simple, with the major effort in the area of address decoding and control signal generation. Fig. 3 shows how this might be accomplished. The methods chosen for simplicity and are based on readily available, inexpensive components. S1 through S4 are address selection switches, which determine the position in the address range the board will occupy. These switches are normally of the DIP-SWITCH type, but may be replaced by jumpers for economy. BOARDSEL-L will be active (low) is all 4 switches match the state of the respective address lines associated with the switches. BOARDSEL-L has several functions, which include enabling the CHIPSEL decoder which is a 7442 decoder, providing an enabling input to both the READ-L and WRITE-L gates and finally in allowing the PROTECT flip-flop (not shown) to be changed by either the PROT-L or UNPROT-L signals. The CHIPSEL decoder is connected so as to provide one of four chip selects according to the state of A10-H and A11-H. Either READ-L or WRITE-L is produced during a memory cycle based on the state of SMEMR-H, PDBIN-H, and MWRT-H lines at the time BOARDSEL-L is low. WRITE-L is used as a memory write pulse and is fed to pin 3 of all the 2102 chips on the board during a memory write cycle. Only those 2102s selected by an active (low) CHIPSEL signal will be written into. READ-L is used to enable the DATA BUS DRIVER which is composed of 8T97 buffers feeding the data-in lines. There are many ways to accomplish the address decoding other than as shown. Among these are the use of address comparator chips to produce BOARDSEL-L.

Keep in mind that the purpose of discussing the memory, CPU, and I/O interfaces is to give an insight into the Altair bus considerations for the boards, not to provide complete design details. With the information provided, it is hoped that you will be able to pick up the ball and make your own designs work. Also, many general designs shown in other articles may be adapted for your Altair, using the techniques in this article.

By far, the largest area open to hardware experimentation in the personal computer system is that of I/O interfacing. If you are started to design an I/O interface from scratch, the first order of business is the conceptual design. This first involved deciding what function the interface board will perform. Once the function has been defined, it must then be decided how the interface will **look** to your software. This is

continued

#	Mnemonic	Description
1	+ 8 Volts	Unregulated power supply for use by +5 Volt on-board regulators.
2	+16 Volts	Unregulated power supply for use by on-board regulators (typically to obtain +12 Volts).
3	XRDY-H	A normally high line, which if brought to the low state will cause the CPU to enter the WAIT state.
4	VI0-H	Vectored Interrupt priority 0
5	VI1-H	Vectored Interrupt priority 1
6	VI2-H	Vectored Interrupt priority 2
7	VI3-H	Vectored Interrupt priority 3
8	VI4-H	Vectored Interrupt priority 4
9	VI5-H	Vectored Interrupt priority 5
10	VI6-H	Vectored Interrupt priority 6
11	VI7-H	Vectored Interrupt priority 7
12-17	Not Used	
18	STADSB-L	Causes the 8 status line buffers on the CPU board to be Tri-stated (enter the high impedance state).
19	C/CDSB-L	Causes the 6 command/control line buffers on the CPU board to be Tri-stated (enter the high impedance state).
20	UNPROT-H	A signal which is ANDed with "board select" on a memory board to cause the PROTECT flip-flop to be cleared.
21	SS-H	Indicates a single-step is occurring in the CPU.
22	ADDDSB-L	Causes the 16 address line buffers on the CPU board to be Tri-stated (enter the high impedance state).
23	DODSB-L	Causes the 8 data-out lines on the CPU board to be Tri-stated (enter the high impedance state).
24	Ø2-H	Buffered TTL compatible version of CPU phase 2 clock.
25	Ø1-H	Buffered TTL compatible version of CPU phase 1 clock.
26	PHLDA-H	"Hold Acknowledge" which is the CPU board response to the HOLD-H input signal.
27	PWAIT-H	CPU signal indicating a WAIT state is occurring.
28	PINTE-H	CPU signal indicating that Interrupts are Enabled.
29	A5-H	Address Bit 5
30	A4-H	Address Bit 4
31	A3-H	Address Bit 3
32	A15-H	Address Bit 15
33	A12-H	Address Bit 12
34	A9-H	Address Bit 9
35	DO1-H	Data Out (from CPU) Bit 1
36	DO0-H	Data Out (from CPU) Bit 0
37	A10-H	Address Bit 10
38	DO4-H	Data Out (from CPU) Bit 4
39	DO5-H	Data Out (from CPU) Bit 5
40	DO6-H	Data Out (from CPU) Bit 6
41	D12-H	Data In (to CPU) Bit 2
42	D13-H	Data In (to CPU) Bit 3
43	D17-H	Data In (to CPU) Bit 7
44	SM1-H	CPU status signal indicating processor is in machine cycle 1 which is Instruction Fetch.
45	SOUT-H	CPU status signal indicating the current cycle is an Output cycle.
46	SINP-H	CPU status signal indicating the current cycle is an Input cycle.
47	SMEMR-H	CPU status signal indicating the current cycle is a Memory Read cycle.
48	SHLTA-H	CPU status signal indicating the CPU is Halted.
49	CLOC-L	A buffered 2 MHz clock for general use.
50	GND	*System ground*
51	+8 Volts	(Same as pin 1)
52	-16 Volts	Unregulated negative power supply for use by on-board regulators (typically to obtain -5 Volts or -12 Volts).
53	SSWDSB-L	"Sense Switch Disable" which is used during an IN 377 instruction to disable the data input buffers on the CPU board so that the sense switches can be "read" by the CPU.
54	EXTCLR-L	I/O clear signal generated by front panel.
55-67	*Not Used*	
68	MWRT-H	CPU signal indicating that the data on the data-out bus is to be written into the memory selected by the address lines.
69	PS-L	"Protect Status" of the selected memory.
70	PROT-H	A signal which is ANDed with "board select" on a memory board to cause the PROTECT flip-flop to be set.
71	RUN-H	Front panel signal indicating that the CPU has been "told" to RUN.
72	PRDY-H	A normally high line which if brought to the low state will cause the CPU to enter the WAIT state. Note: This line is driven by a continuously enabled Tri-state driver on the front panel board and contrary to what others may be doing, this line should *not* be used for any other purpose. The proper line to use for entering wait states by

Table 1. The Altair Bus

the area in which most of the design trade-offs take place. This is to way, for example, if the interface is to be extremely easy to control from the program, then the hardware complexity will likely increase. Conversely, minimal hardware complexity usually results in more difficult programming. This is the real beauty of designing your own interfaces . . . you make the design trade-offs to suit your own needs and tastes.

Many times it is helpful to jot down notes on the way your prospective interface will appear to the software and then make a trial subroutine using the scheme you have decided upon. If the results of your test subroutine are not pleasing to you, then rehash the conceptual design and try again. In this way, you will have a good feel for the way your interface will function before it is built. It will also become apparent as to which way the trade-offs must be moved before trying again.

Three major operating modes for I/O interfaces are **flag testing, program interrupt,** and **DMA** (direct memory access). The most frequently used and easiest to

		"slow" memories and I/O devices is XRDY-H (Pin 3).
73	PINT-L	"Interrupt Request". If Interrupts have been enabled, a low level on this line causes the CPU to enter the interrupt acknowledge condition at the conclusion of the current instruction.
74	PHOLD-L	An input signal to the CPU which causes a HOLD state to occur. PHOLD is the requesting signal for a DMA transfer.
75	PRESET-L	A system reset signal used primarily by the CPU board. (I/O boards normally use the EXTCLR-L signal for resetting).
76	PSYNC-H	A buffered CPU signal which indicates the beginning of a new machine cycle. This signal is used on the CPU board to enable the loading of the system status latch.
77	PWR-L	"Processor Write" which indicates that the data on the data-out bus is to be written either to a memory or an I/O device.
78	PDBIN-L	"Processor Data Bus In" is used to indicate to the selected memory or I/O device that the CPU expects data on the data-in bus.
79	A0-H	Address Bit 0
80	A1-H	Address Bit 1
81	A2-H	Address Bit 2
82	A6-H	Address Bit 6
83	A7-H	Address Bit 7
84	A8-H	Address Bit 8
85	A13-H	Address Bit 13
86	A14-H	Address Bit 14
87	A11-H	Address Bit 11
88	DO2-H	Data Out (from CPU) Bit 2
89	DO3-H	Data Out (from CPU) Bit 3
90	DO7-H	Data Out (from CPU) Bit 7
91	DI4-H	Data In (to CPU) Bit 4
92	DI5-H	Data In (to CPU) Bit 5
93	DI6-H	Data In (to CPU) Bit 6
94	DI1-H	Data In (to CPU) Bit 1
95	DI0-H	Data In (to CPU) Bit 0
96	SINTA-H	Interrupt Acknowledge signal from CPU.
97	SWO-L	CPU status signal indicating that the current cycle involves writing to a memory or I/O device.
98	SSTACK-H	CPU status signal indicating that the address bus contains the stack address and that a stack operation will occur on the current cycle.
99	POC-L	Power On Clear reset signal
100	GND	System ground

Table 1 (continued)

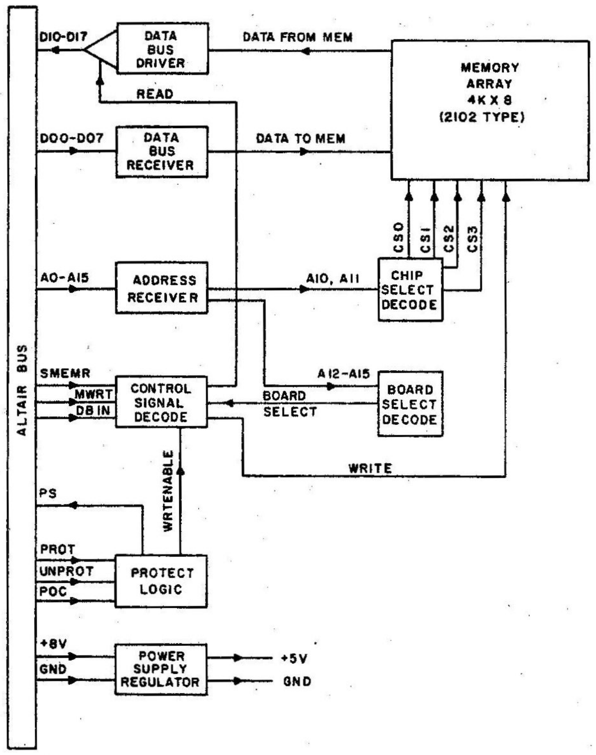


Fig. 2. 4K Static Memory Block Diagram.

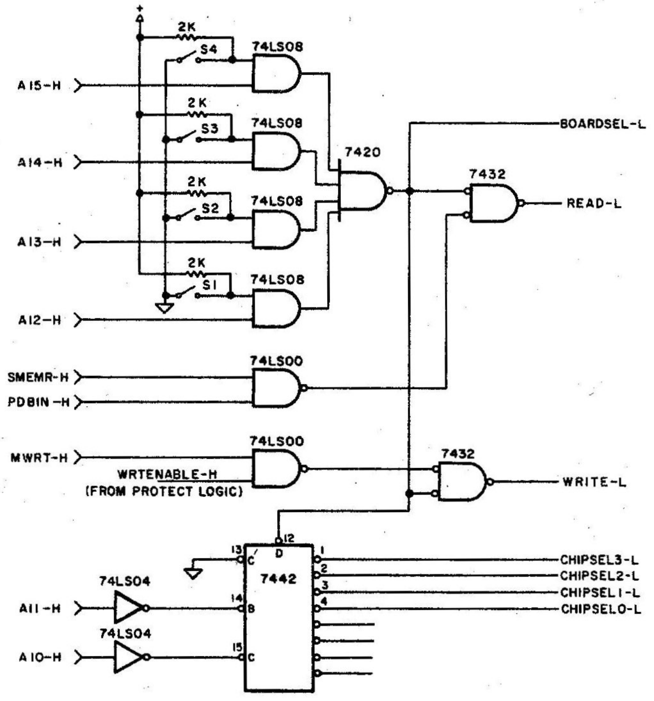


Fig. 3. Address Decoding and Control Signal Generation.

implement) is the flag **testing method.** During the conceptual design, a particular bit is designated to indicate a particular meaning such as **device busy, device ready, device error,** etc. The meanings such bits may assume is limited only by your imagination. Normally, these bits are **read** by the program from an I/O port reffered to as the **status register.** Also, the status register is usually the lowest port number associated with a device. An example might be a paper tape reader in which the status register is port 6. The **data register** would then be port 7. Bit 7 would be a good selection to indicate data ready in the status register. The software would then consist of a **ready loop** shown in Example 1. This type of I/O software is referred to as **flag testing.** Most devices are readily controlled in this manner. The primary shortcoming of the flag testing method is that the computer spends most of its time in the read loop, waiting for the next data to become ready. In most personal computing situations this waiting

continued

BUILD YOUR OWN INTERFACE
Tips from a professional
continued

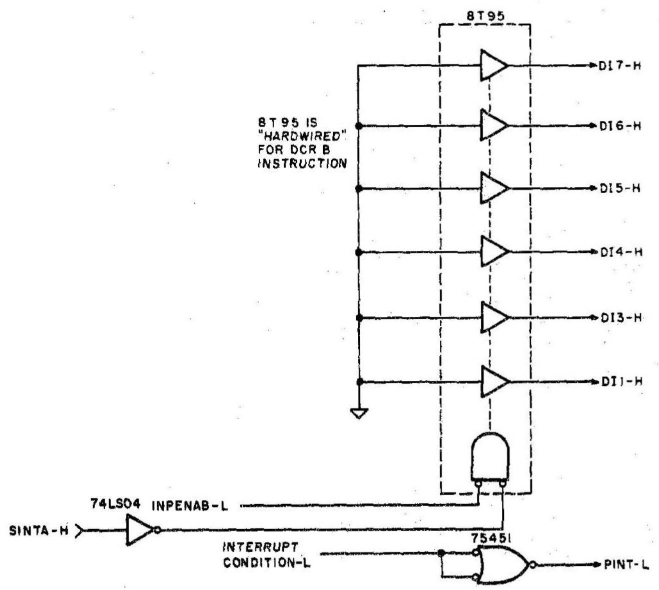


Fig. 4. Interrupt Logic.

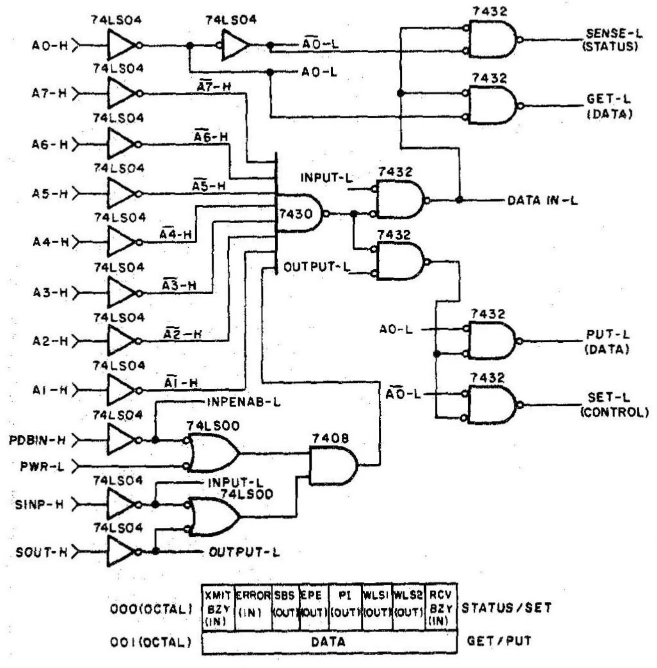


Fig. 5. Port I/O Address Decoding.

```
LOOP:       IN 6      ; GET STATUS
            RAL       ; POSITION READY BIT IN CARRY
            JNC LOOP; TRY AGAIN IF NOT READY
            IN 7      ; GET THE DATA
            MOV M,A   ; STORE THE DATA
            . . . ETC
```

Example 1.

is not a problem. If more than one process must occur simultaneously, it is possible to use program interrupt or DMA to **free** the processor for other processing while the I/O devices function more independently. In the case of program interrupt, the **ready** bit would be used to pull down the interrupt request line (PINT-L). The processor would them respond (if interrupts were enabled) with SINTA-H. The I/O interface would then use SINTA-H to gate an interrupt instrution onto the bus (see Fig. 4). The interrupt instruction is usually a restart (RST) instruction which would save the program counter on the stack and then vector to the I/O subroutine. The I/O subroutine would then process the data and execute a RET (return) instruction. The DMA type of operation will be discussed separately.

Another major decision to be made during conceptual design is whether standard I/O (port I/O) or memory mapped I/O addressing will be used. Each of these methods have advantages and disadvantages which must be weighed in your own mind. Port I/O has the advantage of having less address lines to decode (8) as well as leaving all 65K of the address space available for memory. It has the advantage of being limited to only two types of instructions in transferring data to and from the I/O device (IN, OUT). **Memory-mapped I/O** is a method in which the I/O device is treated as if it were a memory location or group of memory locations. Memory mapped I/O has the advantage of being able to use any of the transfer instructions that are used with real memory, including arithmetic and logical as well as move instructions. The disadvantages are the need to decode more address bits (16) and the fact that part of memory address spectrum is consumed for I/O use. An example of port I/O address decoding is shown in Fig. 5. An example of memory mapped I/O is given in Fig. 6. A bit chart with addresses is provided in each of these figures. Each is set up to perform an identical function so that you may compare the complexity of one to another.

The next step in the design of the I/O interface is the design implementation. This is the part which is referred to by many people simply as the design stage. This is the actual drawing of the logic diagram in such a way to satisfy the requirements of the bus (referred to as bus overhead). To aid you in this area a typical I/O interface block diagram is provided in Fig. 7. Also virtually every I/O interface requires the sending of data to the CPU board. A means of performing this function is shown in Fig. 8. Fig.8 depicts a method in which data and status are multiplexed into the data buffers. Other methods of accomplishing this include separate 8T97s for the data and the status; thus, eliminating the 74157 multiplexers.

When designing an interface which makes use of large scale integration (LSI) components, the control register, status register, and device logic are all **built in** to a special purpose chip. Examples of this type of chip include the UART, PIA, ACIA, Programmable Peripheral Interface, etc. These chips are very useful because they were designed for microprocessor (or minicomputer) interfacing ease and package count efficiency. When using these devices, the specifications and application notes should be studied carefully, especially in the areas of operating modes, software considerations and performance. In the specification sheets, beware of using the **typical** values. Instead, use the minimum and maximum values as appro-

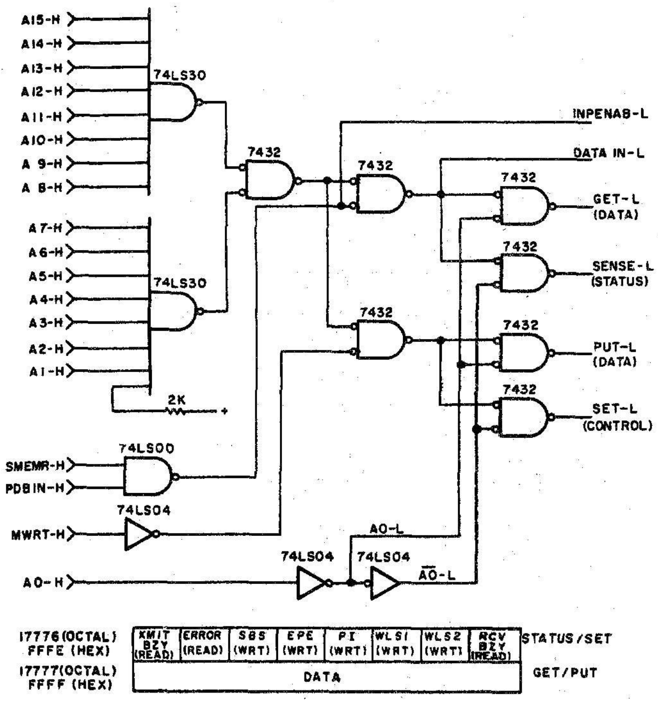


Fig. 6. Memory – Mapped I/O Address Decoding.

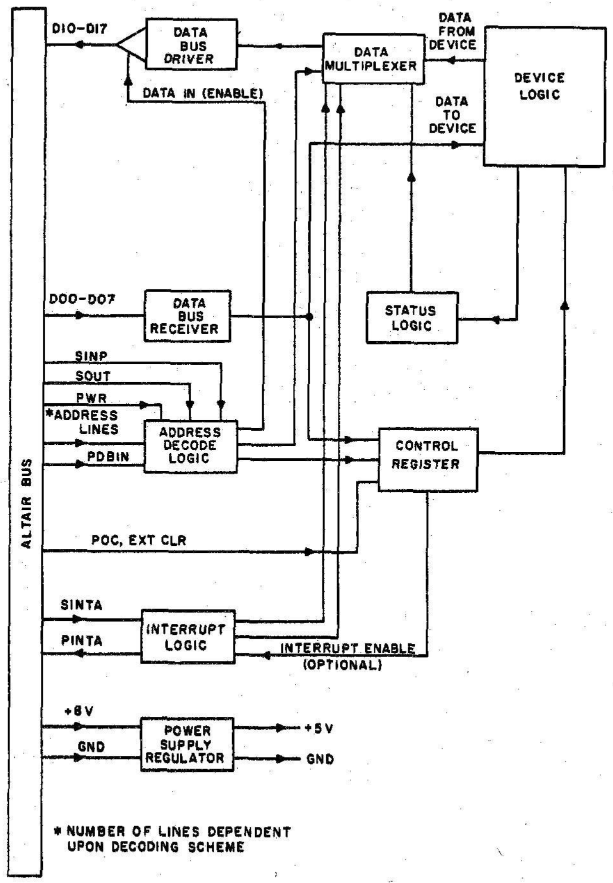


Fig. 7. Typical I/O Block Diagram.

priate. This will help to assure glitch-free operation of your finished interface. Since most of these LSI components are MOS (Metal oxide semiconductor) chips, careful attention **must** be paid to output signal loading and timing specifications. Almost without exception, these devices can drive only one TTL load and should therefore be buffered. The 8080 MOS/LSI microprocessor chip itself is a good example of this fact. Notice that all output lines of the 8080 are buffered on the CPU board. Also most of these devices must be fed relatively wide timing pulses. In addition, some MOS/LSI devices are dynamic in nature and thus require a constant clock of some minimum frequency. These requirements are all relatively easy to meet, but overlooking them will, in most cases, cause disappointment in the performance of your interface. While these devices are more difficult to use than standard TTL integrated circuits on a chip for chip basis, the large number of chips and the printed circuit space they replace make them very worthwhile additions to an interface design. Also, these devices use far less power than the group of TTL chips they replace. As in the handling of all MOS components, care must be taken in the handling of these devices as they are subject to damage from static charges.

Since the control register and status register are usually incorporated internally in the LSI components, the conceptual design must conform to the available control and status bits provided. In some cases, it is possible to augment the control/status of the LSI chip being used by providing additional external flag bits or status bits. An example of this would be an **interrupt enabled** control bit external to a 1402 UART (which has no such function internally). It is also possible in some cases to combine outputs (by using external gating) in such a way as to make the signals more useful to your unique application. In other cases, it is possible to ignore certain outputs if they are not suited to your needs. On the unused input signals, it is usually necessary to **tie** the pin high or low or to another driven input. Carefully review the specifications of the device for clues as to the handling of unused inputs. It is easy to be intimidated by the maze of data and buzz words contained in many manufacturers' data sheets. Keep in mind that reading these sheets does **not** require a degree in engineering and that the people preparing the data sheet **want** you to understand it. Also, remember that data sheets **do** sometimes contain errors, so if in doubt, check with the manufacturer or his representative (usually given on the back of the data sheet) or check on the data from a different vendor if the part is second sourced.

As mentioned earlier, it is possible to free the processor of much of the work involved in I/O programming by using direct memory access (DMA). Fig. 9 shows the block diagram of a DMA controller. The basic operation of the controller involves setting the **DMA address register** to the desired number of bytes to be transferred and then allowing the DMA controller (independent of the CPU) to transfer an entire block of data to (or from) the device from (or to) the main memory. When each byte is transferred, the byte counter is decremented while the address counter is incremented. When the byte counter reaches a count of zero, the DMA operation is complete. The DMA transfer consists of a number of a DMA bus cycles which is equal to the number of bytes specified

continued

when setting the byte counter. Each DMA cycle is initiated by the device requesting service, which results in HOLD-L being driven low (active). This is then recognized by the CPU board which sychronizes HOLD-L and eventually suspends program activity and responds with HLDA-L. When the DMA controller receives HLDA-L, the **disable** signals STADSB-L C/C DSB-L, ADDDSB-L, and DODSB-L are all brought low which removes virtually all CPU influence from the bus. The DMA controller then drives the disabled signals in such a way as to make the other boards in the system (especially the memory boards) **think** that the CPU is **talking** to them. The direction of the data transfer, the memory address, and all other control functions are determined by the way these lines are driven.

Among the advantages to be gained by using the DMA facility are transfer rates which greatly exceed the maximum programmed data rate (I/O under CPU control). The number of integrated circuits required to implement DMA is rather large, which tends to discourage its use in all but the most demanding situations. In most cases, due to the large number of components involved, the divice logic and the DMA logic are contained on separate boards and are interconnected externally from the bus to one another by means of jumper **straps.**

While not one of the more glamorous apsects of computer interfacing, the need for proper power distribution remains ever present. The majority of the logic used in the Altair system requires the use of 5 volts which must be regulated to within 5% of nominal. The bus supplies unregulated positive voltage on pins 1 and 51 at approximately 8 volts. Each board then regulates this down to 5 bolts for use on that board. The easiest way to accomplish this regulation is through the use of one of the 3-pin regulators which are available in several ratings and shapes. The two most common types are the 7805 and LM309 regulators. Most commonly, the 7805 comes in the TO-220 package while the LM309 uses the TO-3 package. Either type requires the use of an adequate heat sink as well as input and output filtering. Both types are nominally rated at 1.0 Amps. For current requirements in excess of 1.0 Amps, multiple regulators may be used, or heavier duty types may be used, such as the LM323 which is rated at 3.0 Amps. Also, it is important to bypass the regulated 5 volt line to ground at regular intervals using .01 uF ceramic capacitors. A good rule of thumb if one .01 uF capacitor for each six integrated circuits. Also, follow the manufacturer's guidelines for any special layout requirements. This is especially important in the area of analog interfacing. Don't overlook the importance of good grounding techniques when constructing your interface as deficiencies in this area sometimes show up as ghosts (erratic operation) and rarely can be found by logical means.

Interfacing is a key ingredient in making your computer perform useful functions as well as an interesting part of personal computing. Becoming skilled in the art and science of interfacing will pay many dividends in the enjoyment of this most interesting hobby.

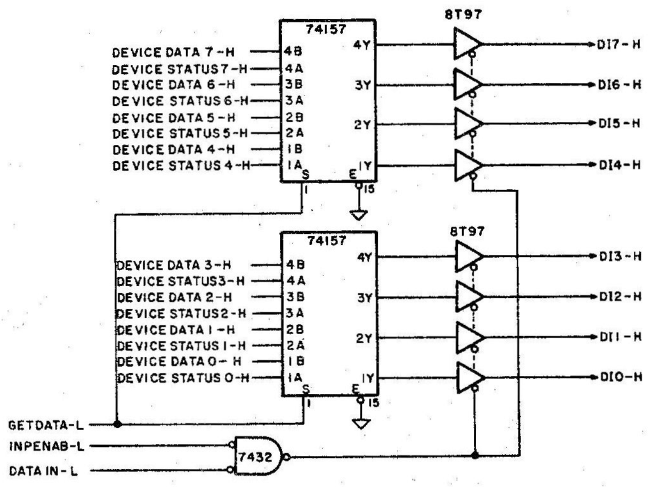


Fig. 8. Data-In Driver/Multiplexer.

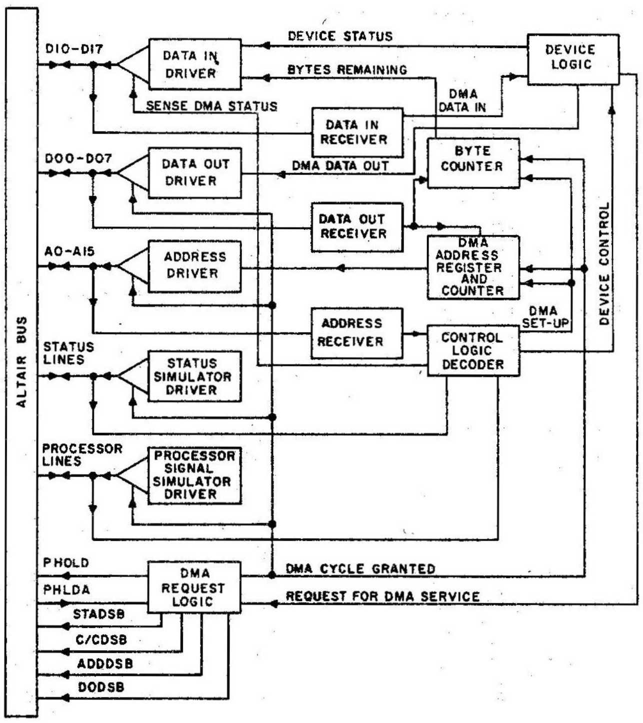


Fig. 9. DMA Block Diagram.

Students Develop IC Logic Test Clip for Altair 680b

By James Gupton
7416-G Pebblestone Drive
Charlotte, N.C. 28212

Gupton is a free-lance writer and an electronics teacher at the Union County Career Center in North Carolina. This is the second in a two-part series of articles on his students' experiences with an Altair 680b computer. (For Part I see "Students Find Altair 680b Kit Easy to Assemble," COMPUTER NOTES, May 1977, pp. 12-14.)

In Phase II of the Altair 680b kit assembly, my students developed an IC logic test clip for the final electrical check-out and error detection before applying power to the system.

First, they carefully examined each subassembly of the 680b kit---the power supply, the front panel and the main board. The assembly checkout revealed two microscopic solder bridges on the main printed circuit board. Apparently, the low-temperature soldering irons have a tendency to lift hairline threads of solder when removed from the soldering point. The remaining checkout showed professional workmanship and was a credit to the students.

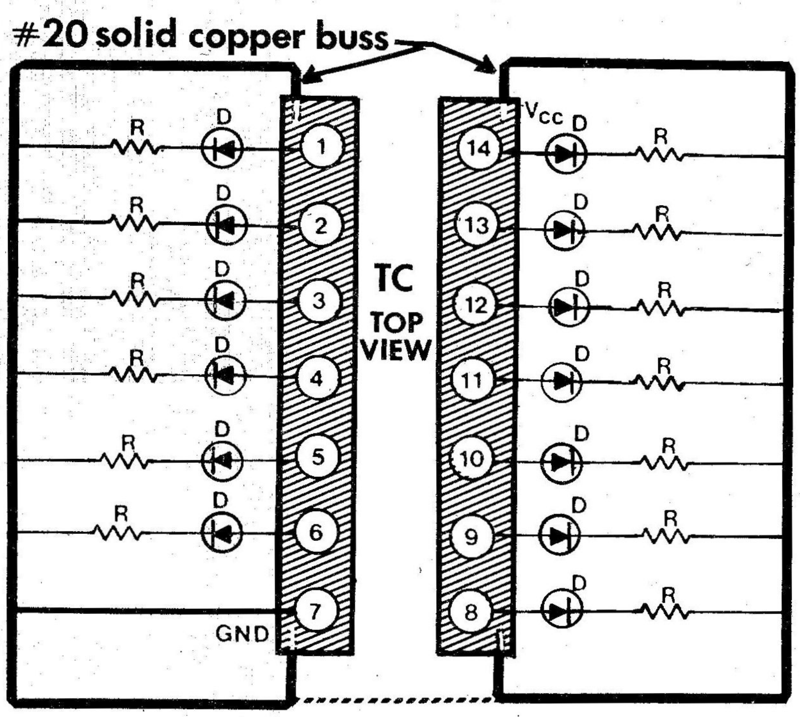
Although two defective SPDT front panel switches had to be replaced, students went ahead with the next step and checked the circuit voltage and logic state on the front panel and main board. The power supply and voltage regulating circuits were fine. Front panel LED's turned on as switches were flipped from zero condition to one condition. Everything seemed to function according to the assembly manual. But students still wondered if they had the proper logic conditions on the logic-integrated circuits. Have you ever tried to juggle a schematic diagram, position and hold a printed circuit board, handle two snake-like meter probes, keep one eye on the meter and keep a wandering test probe on a specific pin of an IC -- all at the same time? It's difficult for even expert computer technicians, so my students had a number of problems. What they needed was an inexpensive "student-proof" device which would tell the logic state and the input voltage of an IC without a meter.

AP Products, Inc., Painesville, Ohio and Radio Shack National Headquarters, Fort Worth, Texas donated 14-pin and 16-pin IC test clips. Radio Shack also donated subminiature LED's and an ample supply of one-quarter watt resistors. Students only had to come up with the idea and assembly time for an IC logic test clip.

Of course, nothing always goes as planned. Although we received the IC test clips and resistors, the LED shipment was delayed, and we received only some of them before school closed for the summer. However, students did complete the 14-pin IC test clip, and with the exception of the LED's they also wired the 16-pin logic tester.

Students felt that the schematic drawing and assembly illustration included with this article would help other Altair 680b owners who want to construct a similar logic IC test clip.

Unfortunately, the school year ended before students could begin writing programs. But that will be one of the first projects for the next school term. Hopefully, we will then be able to get a keyboard interface and a cassette tape interface to really make our Altair 680b hum. . . . . er. . .compute!



D = LED (Radio Shack 276-042)
R = 150 OHM 1/4 Watt Resistor
TC = IC TEST CLIP (Radio Shack 276-1950 14pin
-1951 16pin

What's a Microcomputer Class without an Altair 8800b?

By Linda Blocki

Two trade schools in Albuquerque recently began offering microcomputer courses using Altair 8800b computers as teaching aids.

Throughout the year, North American Technical Institute (NATI), which teaches only electronics and math, offers "Microcomputer Circuitry (361)," and Albuquerque Technical Vocational Institute (TVI) offers "Digital Circuits IV" to students with a basic knowledge of electronics. The courses are part of each school's Associate of Science degree in Electronic Engineering Technology.

Both schools use their Altair 8800b with 5K bytes of memory to teach students how to run the system, program in machine language, and interface with various I/0 devices. Assembly is not covered in either class. However, each class emphasizes different aspects of microcomputing. NATI Director Roy Stone said his school's main goal is employment security. "So whatever we teach is designed to help students get a job as an engineer or technician in the computing field," he said. "We're very responsive to students' needs."

NATI instructor Dennis Crunkilton said since the school is primarily hardware-oriented, theory, design, and troubleshooting make up the majority of the course. "Students are taught only a minimum of programming which will allow them to understand the hardware workings of the 8800b," he explained. NATI does offer a separate course on "8080 Machine Language" and is also planning more software classes (BASIC and Fortran) in the near future. Stone said those classes will begin as soon as the school orders several more Altair computers with 32K bytes of memory and five more terminals. "Although we'll offer both BASIC and Fortran, we really prefer Fortran because so many of our graduating students work with scientists," he added.

Crunkilton said the decision to use an Altair 8800b for the class was due, in part, to the new Altair Timesharing BASIC. He said its cost-effectiveness really sold him on Altair microcomputers. "With minicomputers, you'd have to buy many terminals. A few years ago, I would have laughed if anyone had even suggested a timesharing system could be set up with microcomputers." But now, by using

Computer Notes Assistant Editor, **Linda Blocki**, discusses microprocessor functioning with TVI instructor, **Cecil Lennox**.

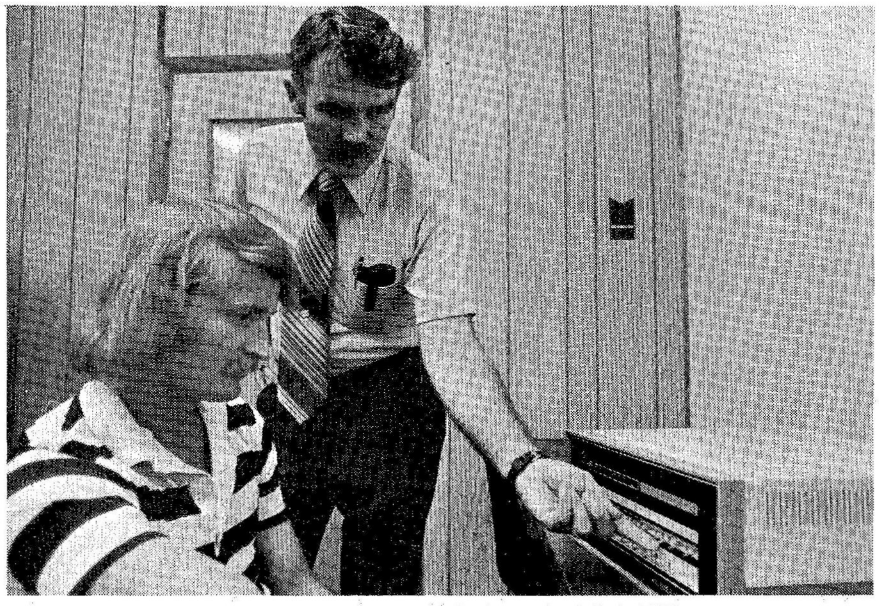
Altair-compatible options, Crunkilton said the capabilities of Altair microcomputers can be expanded to that of minicomputers. "So I can teach students about minicomputers on our Altair 8800b. This is just one of the advantages microcomputers have over minicomputers," he said.

TVI instructor Cecil Lennox said in addition to working with the Altair 8800b, his students study the circuitry on an 8080-based microprocessor with 256K bytes of memory and also work with a PDP/11 minicomputer. Troubleshooting is not covered in the class.

Lennox said TVI plans to expand the course to a full-time digital program that includes troubleshooting and programming in BASIC, machine language, Fortran, and Cobol. He said in the fall TVI will bid on three new systems with 8K and 16K bytes of memory and video terminals to use in these classes.

A 15-week night class aimed at people in the electronics/minicomputer industry will also be offered at TVI this fall, he said. "Students will develop their own plans for a microprocessor and then build a microcomputer with 1K bytes of memory," he explained. "The class will also include troubleshooting. It will be good hands-on experience," he said.

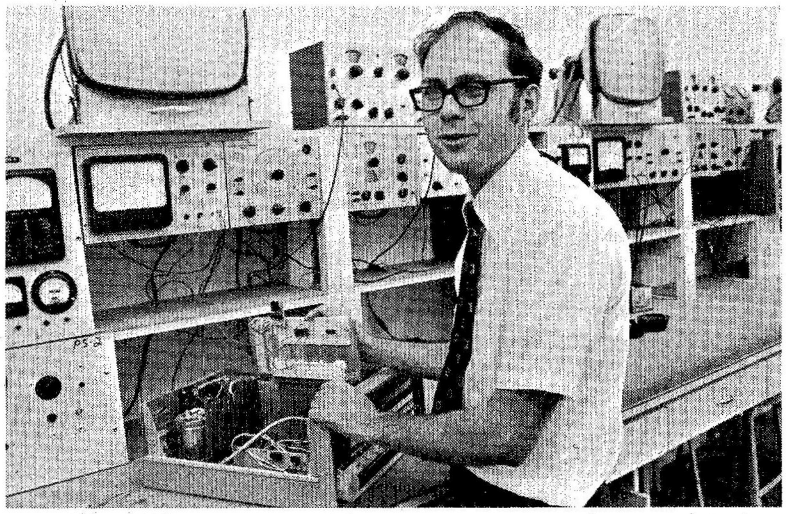
The two instructors and Stone said

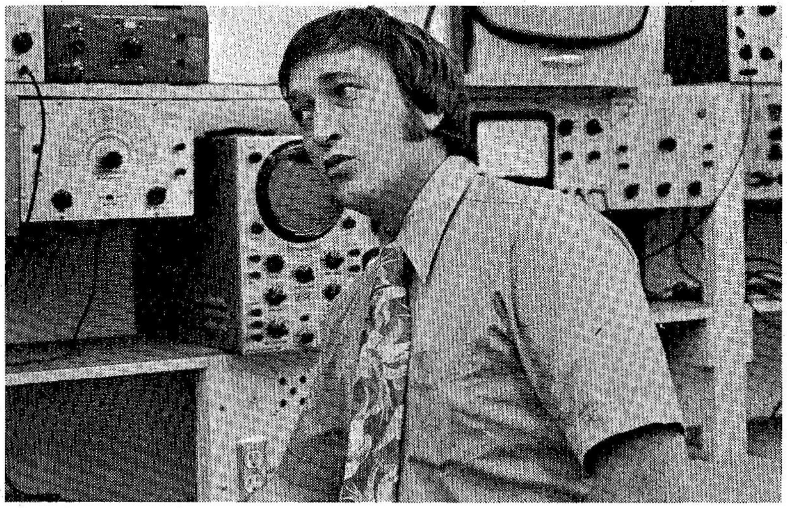

TVI instructor, **Cecil Lennox,** shows one of his students fundamentals of **Altair 8800b** operation.

Beginning in October, Lennox will also teach an eight-week microcomputer course at MITS. Geared for people with a non-technical background, the main goal of the course is to provide an overall picture of an integrated computer system, including the fundamental anatomy of a computer, the mainframe, peripherals, system hook-up, and software.

capable students with electronics experience don't have any problem finding a job just about anywhere in the country. "Most students are working even before they graduate," said Stone. "Because so few schools in the U.S. teach our in-depth level of electronics, there's a great demand for industrial instructors, field engineers, and technicians." Stone said this demand can't be met by private industries or public universities. "Neither can offer the necessary in-depth approach to electronics. Universities don't set up such a program because they need to turn out well-rounded students," he explained. One solution Stone suggested is that industries encourage employees to attend electronics schools and pay for their tuition.

With the current rapid expansion of the electronics field, even instructors must keep up-to-date with the many new changes. This sometimes means going back to school. In order to learn more about microcomputers for his "Digital Circuits IV" class, Cecil Lennox spent many half-days at MITS this summer studying the Altair 8800b schematics and circuitry and various I/0 devices. Lennox said learning about I/0 devices was more difficult and required a great amount of concentration. But he said I/0 devices are an important part of his class at TVI. "The industry is particularly demanding people who have training in I/0 devices to control industrial processes," he said.




NATI Director, **Roy Stone,** says electronics students have little trouble finding jobs after graduation.

Programmable I/O Made Possible with the PIA

By Dave Antreasian

The PIA -- magical black box with multi-functions, a baffling name and as many pins as an 8080 chip. How is it used and why?

Before answering these questions, it's important to realize that the days of simple, discreet component circuitry are quickly vanishing. Since it's obviously easier and less costly to produce a product with only a few parts, the current trend is toward higher integration assemblies. However, IC manufacturers cannot be expected to produce a specialized chip for the millions of diverse electronic products on the market today. For this reason, highly condensed yet extremely versatile IC's like the PIA will become increasingly popular in the future. However, with this increased versatility, designers will have to provide more control information (especially more software control) to define functional conditions to the chip. Hopefully, this article will clear up some of the mystery and confusion regarding initialization of the PIA.

6820 PIA

The (6820) peripheral interface adapter is actually a highly condensed data bus buffer system complete with handshake lines. Used on several of the 8800-based MITS cards (Altair 88-ADC, 88-AD/DA, 88-4PIO, 88-PCI) and 680 cards (Altair 680 Universal I/O, 680-PCI, 680-AD/DA), the 6820 contains enough circuitry to replace approximately 10 standard TTL IC's.

Functions of the 6820 include two independent sections -- A and B. Each contain:

1. Eight data lines which can be independently set up as inputs or outputs (or both be reinitializing during system usage and multiplexing from one function to the other).
2. An independent input (Flag) line CA1 or CB1) with presetable transition polarities (also enabling or disabling respective interrupt request (IRQ) lines).
3. An Input/Output line (CA2 or CB2) which can be set up as either an input flag similar to CA1, CB1 or used as an output line to strobe external circuitry. If used as an input, this flag is independent of the CA1, CB1 flag. If used as an output, this line can be set directly by writing a command word to the Status Register or by a READ/WRITE command to the DATA Register(WRITE for Section B, READ for Section A). By re-initializing during system usage, the Input/Output functions of this line can be multiplexed. (See Fig. 1 for a block diagram of the IC.)

4R/W and "E" lines

The R/W line is used by the PIA to define either a READ data command from the PIA or a WRITE data command to the PIA. A low-to-high transition of the "E" line is used to strobe data into the PIA or latch data to be read by the CPU. Note that when writing to the PIA, data is latched onto data lines defined as output lines but is not latched when such lines are treated as input lines.

Reset Line

The reset line is used to clear all internal registers, set up data lines as inputs and set up CA2, CB2 lines as inputs with IRQ lines disabled. Initialization software must redefine all functions after a reset is applied.

Register Selection

There are three internal registers in each section:

1. Status Register -- used to specify transition polarities of flag lines and enable interrupts.
2. Direction Register -- used to specify direction of data flow for each of eight data lines. (Useful during initialization sequence only.)
3. Data Register -- accessed when reading from or writing data to the PIA.

Usually, the Direction Register will not be accessed except during initialization. It has the **same** channel address as the Data Register and is accessed by writing a (O) into bit 2 of the Control (Status) Register. Once this is done, all calls to the Data Register Ch. # will access only the Direction Register, **not** the Data Register. Therefore, the following sequence of initialization is required for each section of the PIA:

1) Access status channel with bit 2 (O)
2) Access data channel (now the Direction Register) and define eight lines as inputs (O) or outputs (1's)
3) Access status channel and define interrupt flags, CA2, CB2 outputs, etc., with bit 2 (1)
4) Next, access to data channel will access Data Register.

Now repeat this sequence for the other section.

After this has been done, initialization is complete, and unless multiplexing of functions is required, further register access will be made to either the Data Register (Read or Write) or the Status Register (Monitor Flags, strobe output lines, etc.)

The flag lines can be used by monitoring the proper status bit (bit 7 for Section 8; bit 6 for Section B) in software. When the required transition (as specified during initialization) occurs, the appropriate register bit will be set high. If enabled (as specified during initialization), the respective IRQ line will go LOW.* If both CA1 and CA2 lines are defined as input flags, the proper transition at either will cause the IRQA line to go low. The IRQ lines may be tied to vectored interrupt bus lines if desired. To clear the interrupt, a read command to the appropriate Data Register must be performed. (See Fig. 2 for a condensed table of functional initialization options.)

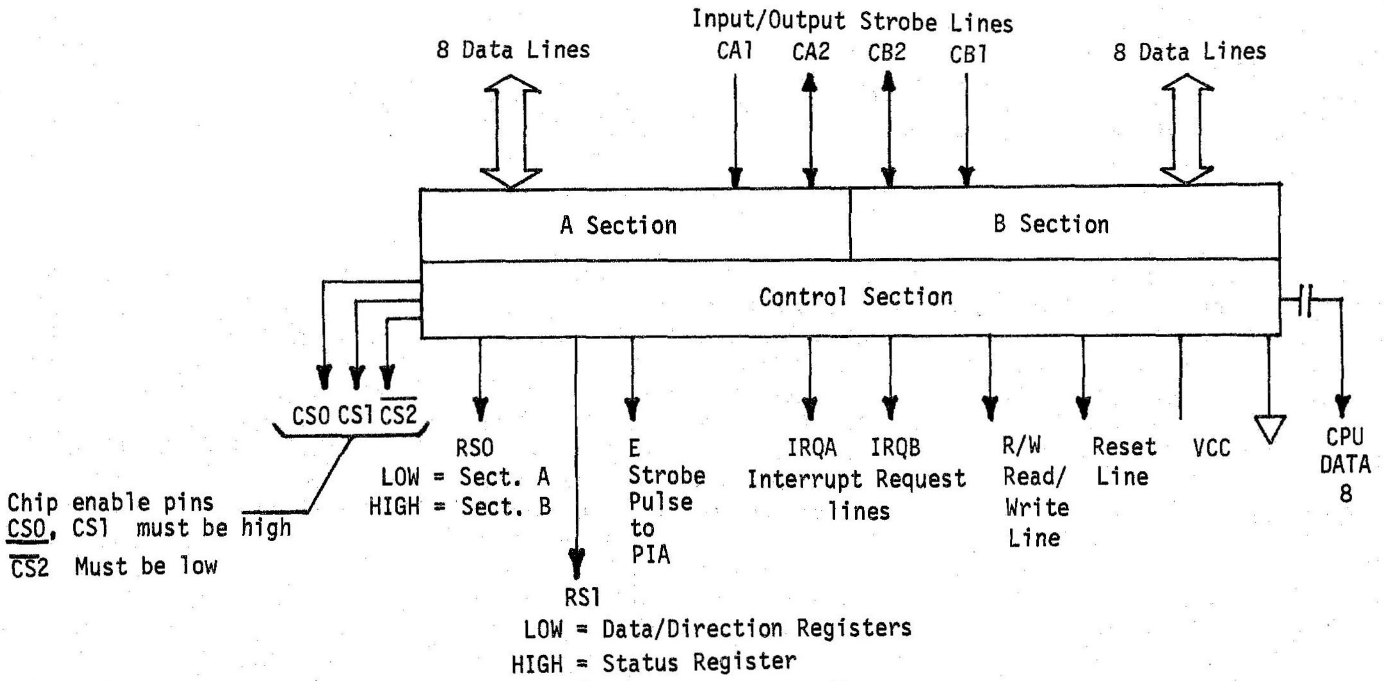


Figure 1.

	SETTING SPECIFIC STATUS WORD BITS	IRQA FLAG	IRQB FLAG	CA2 (OR CB2) SETUP			DATA/ DIRECTION ACCESS	CA1 (OR CB1) SETUP	
SETS		Bit 7	6	5	4	3	2	1	0
UP	CA ↓ SETS IRQ **FLAG** ↑, IRQ DISABLED	X	X	X	X	X	X	0	0
CA1	CA1 ↓ SET IRQ **FLAG** ↑, IRQ ENABLED	X	X	X	X	X	X	0	1
(OR CB1)	CAL ↑ SETS IRQ **FLAG** ↑, IRQ DISABLED	X	X	X	X	X	X	1	0
TRIGGER	CA1 ↑ SETS IRQ **FLAG** ↑, IRQ ENABLED	X	X	X	X	X	X	1	1
INIT.	DEFINES DATA CH. ADD AS DIREC. REG.	X	X	X	X	X	0	X	X
SETUP	DEFINES DATA CH. ADD AS DATA REG.	X	X	X	X	X	1	X	X
SETS	CA2 ↓ SETS IRQ **FLAG** ↑, IRQ DISABLED	X	X	0	0	0	X	X	x
UP CA2	CA2 ↓ SETS IRQ **FLAG** ↑, IRQ ENABLED	X	X	0	0	1	X	X	X
(OR CB2)	CA2 ↑ SETS IRQ **FLAG** ↑, IRQ DISABLED	X	X	0	1	0	X	X	X
TRIGGER	CA2 ↑ SETS IRQ **FLAG** ↑, IRQ ENABLED	X	X	0	1	1	X	X	X
SETS	CB2 ↓ FOLLOWING E ↑ 1, CB2 ↑ WHEN BIT 6 ↑	X	X	1	0	0	X	X	X
UP	CB2 ↓ FOLLOWING E ↑ , CB2 ↑ ON NEXT E ↑	X	X	1	0	1	X	X	X
CB2	CB2 SET ↓ DIRECTLY FROM STATUS WRITE	X	X	1	1	0	X	X	X
OUTPUTS	CB2 SET ↑ DIRECTLY FROM STATUS WRITE	X	X	1	1	1	X	X	X
SETS	CA2 ↓ FOLLOWING E ↑ 2, CA2 ↑ ON BIT 7 ↑	X	X	1	0	0	X	X	X
UP	CA2 ↓ FOLLOWING E ↑ , CA2 ↑ ON NEXT E ↑	X	X	1	0	1	X	X	X
CA2	CA2 SET ↓ DIRECTLY FROM STATUS WRITE	X	X	1	1	0	X	X	X
OUTPUTS	CA2 SET ↑ DIRECTLY FROM STATUS WRITE	X	X	1	1	1	X	X	X

NOTES

X Don't Care

´ A Low-To-High transition signal

´ A High-To-Low transition signal

1 CB2 on the next E following a WRITE command to the Date register

2 CA2 on the next E following a READ command to the Data register

Figure 2.

*Similar to an open-collector output-pull up to Vcc, Active LOW.

* CA1 & CB1 are inputs only

* CA2 & CB2 can be inputs/outputs

* IRQ FLAGS are interrupt flags set only by input lines as setup in initialization (they cannot be set by software). They are always reset by a data read from the appropriate section.

* Trigger transition selected for flag lines will always set status bit flag regardless of IRQ ENABLE/DISABLE. If flag is set and interrupts are then enabled, IRQ line will immediately go active (LOW).

Note that sections (A) and (B) are identical **except** when using CA2 and CB2 as outputs

GLITCHES
Troubleshooting the 88-4PIO

By Bruce Fowler
MITS

Using the Altair 4PIO board permits the input and output of data in parallel form. Unlike the serial 2SIO or ACR boards, the parllel data has no framing (start and stop bits) associated with it. The serial start bit tells the ACIA or UART that new data is being entered. Parallel data may change several times before the valid data is stable, so some means is required to tell the 4PIO when valid data is present.

This can be done by a handshaking signal to either CA1 or CB1 (and sometimes CA2 or CB2). An active transition on the CA1 or CB1 level causes bit 7 of the appropriate control register to go high. So if bit 7 of the Control Register is monitored, the CPU can tell when new valid data is ready to be input. While the CPU inputs this new data, the output terminal must keep it stable. This input data is not latched into the 4PIO chip.

Inputting on a 4PIO can be compared to an enabled buffer. When the CPU has input the data, bit 7 of the corresponding Control Register is cleared to zero, thus denoting that no new data is waiting to be input. Another handshaking signal may be used to tell the output terminal that more data can now be sent. Note also that unlike the UART and 2SIO, no error detection takes place. The 4PIO chip (the 6820 PIA) does not generate parity or check for overrun. Overrun error occurs when data is sent before the previous data has been read by the receiver. It is the responsibility of software and hardware external to the 6820 to avoid overrun errors.

By the same reasoning, the 4PIO must tell the output terminal when new data is ready for it. This is again done by handshaking signals, usually CA2 or CB2. These tell the output terminal when new data exists. Unlike parallel input, data to be output is latched into the 4PIO chip. This allows the 4PIO to keep outputting the same data while the CPU does other tasks. Otherwise, the CPU would have to wait until the output terminal indicated that it has strobed in the data.

As with the ACIA (on the 2SIO board), the 6820 must be written into or initialized by software. The immense variety of combinations may be confusing to the user. If the 4PIO Theory of Operation Manual is inadequate, see "Software Initialization of Parallel and Serial I/O Boards," by Pat Godding, June 1976, CN or "4PIO Operation," by Bill Kuhn, October 1976, CN. The best source for specs is M6800 Microcomputer System Design Data from Motorola. Osborne's INTRODUCTION TO MICROCOMPUTERS, Vol. 2, SOME REAL PRODUCTS also has an explanation of the 6820. (Please note the error on p. 6-48 of this book. Bits 6 and 7 of the Control Registers are cleared by a read on the Data Register and <u>not</u> on the Control Register.) In spite of this error, the book explains the 6820, especially handshaking. I used the interrupt input and output handshaking configuration as described in this book to check the 4PIO.

TROUBLESHOOTING

Problems with the 4PIO can occur in several areas: wait circuitry, selecting and control circuitry, and the data lines. Most problems can be isolated by single stepping an input (IN) or an output (OUT) instruction on the data or control channel. Readers with an Altair 8800B should not use the M1 single step option for the following. The following deals with the first port of a 4PIO addressed at location 40 (octal). Users should be able to alter the following explanation so that it applies to their particular case.

WAIT CIRCUITRY

The 6820 was designed for a slower running microcomputer (Motorola now manufactures a faster 6800 and family). A wait state is required for inputting data from the 6820. To check the wait circuit, run the prgram in Table 1. If low-going pulses occur at pins 13, 14 and 15 of IC C, then the wait circuitry is working. If not, check pin 2 of IC 0 for low pulses. If no pulses are present, trace logic back to PWAIT (which should be pulsing on the bus) and to POC (which should always be a high level) on the bus. Repair is necessary. If this is not the problem, compare levels while the program is running to column A of Table 2. Trace logic back to the bus for any discrepancies. Check the clock to IC 0 by stopping the computer and examining to location 000. Compare the levels on the 4PIO board to column B of Table 2. Press single step two times and monitor pin 13 of IC 0 for a low pulse. Compare levels to column C of Table 2. Repair as necessary.

CHIP SELECT AND REGISTER SELECT

When inputting or outputting to a 6820, all three chip selects must be active or no data transfer takes place. There are four addressable registers within the 6820 -- two for section A and two for section B. RSO and RS1 select one of the four.

CONTROL SIGNALS

One of the three control signals, RST, simply clears the 6820 when the chasis is powered up. R/W is the opposite of SOUT. If an output instruction is executed, SOUT will go high, causing a low R/W. R/W controls the bi-directional bus direction of the 6820. All timing is centered around the enable signal, which is derived from the logical OR of PWR and PDBIN. PWR is active when an output instruction is executed, and PDBIN is active when an input instruction is executed.

CHECKING DATA LINES AND CONTROL SIGNALS

To test the data lines as well as the above mentioned signals, do the following. Connect a wire to ground, perhaps the chassis itself. Toggle in the program in Table 3 up to location 030. For the following, press single step the given number of times. Check the TTL levels and repair as necessary. Then continue on single stepping from the point left off in the program. Examine to location zero. Press single step four times and compare levels to column A of Table 4. Press single step three more times and compare levels to column B of Table 4. Press single step three more times and compare levels to

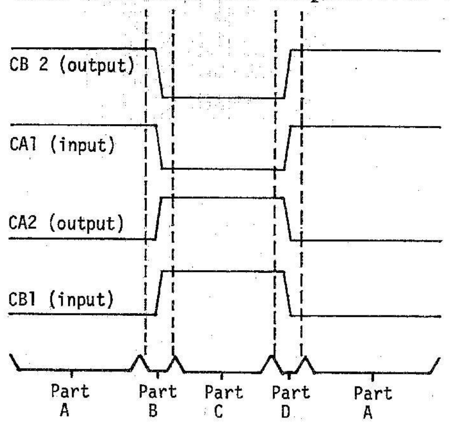


Figure 1 - Handshaking Signals

column C. Press single step five more times; compare to column D. Press single step 13 more times and compare to column E.

At this point data flows straight from PA0-PA7 to data bus lines DIO-D17. The data shows up on the data lights. Since nothing is connected to PA0-PA7, all highs should appear on the data lights. TTL logic, which is what section A expects, interprets open inputs as highs. If one or more of the data lights do not light, trace the appropriate data line from the 6820 to the bus, looking for shorts. Repair any bad ICs when found. Take the ground wire and connect it to PA0-PA7 one at a time. By doing this with the 4PI0 cable inside the circuitry, it will also check the cable. (See Table 5.) Be extremely careful not to short any two points together. As each PA data line is grounded, the corresponding data light on the front panel should go out. Otherwise, trace the logic and repair as necessary.

ECHOING

To check the 4PI0 data lines fully, an echo is required. Don't rely on the output terminal, since it may or may not be functioning. Thus, an echo plug is required. Take a DB-25 male pin connector and wire it according to Table 5. If one is not available, wire the corresponding pins of the 6820 socket together as shown in Table 6. You may want to take out the 6820 to avoid overheating it. This wiring simply connects section B (PB0-PB7, CB1, CB2) to section A (PA0-PA7, CA1, CA2). The program in Table 3 initializes the 6820 so that section B, which can drive more current, is output, and section A is input.

Toggle in the full program in Table 3. It will be used later for testing handshaking, and is thus extensive. Run the program. Stop the computer. If the address lights show address bit A7 high, then the 4PIO echoes data correctly. Go on to the next section. If the address lights show address sit all high, then a transmission has occurred. The byte that was output is stored in location 000101. The byte that was input is stored in location 000100. Find which bits do not match, and trace the logic for the corresponding data line from the bus to the 6820 and around the echo plug to the 6820. (The input data line has already been checked). Look for opens and shorts on the data lines. If another 6820 is available, replace the 6820 with it. Notice that when data is output from the 6820, from the next enable pulse on until new data is output, that data will be continually present on the output pins. Output operation can be checked by measuring the levels on the assigned output pins of the 6820 and knowing what was output. An input will temporarily tri-state section B data lines if they are assigned as outputs.

HANDSHAKING

Handshaking signals are provided to announce the presence of valid data. To check the operation of the handshaking signals, put in 037 at location 044 and 055 in location 062 in the program of Table 3. Examine to zero. Press single step 37 times. Monitor CB2 and CA2. Press single step three more times. As shown in Figure 1, CB2 should go low and CA2 should go high. Single step two more times. Bit 7 on the data lights should be high, indicating that bit 7 of Control Register A is high. Otherwise, check for shorts. Single step nine more times. CA2 should go low, while CB2 should go high. Single step seven more times. Bit 7 on the data lights should be high, indicating that bit 7 of the B Control Register is high. Otherwise, check for shorts or opens on the CA2 or CB1 line. The low-to-high transition of CA1 (or CB1) sets bit 7 of Control Register A (or B). Single step 17 more times to complete one full loop. Make sure you are at location 035. Repeat this procedure one more time (starting with single stepping two more times). This should be enough to check your 4PIO out for handshaking.

Figure 1 is useful for understanding the 4PIO handshaking. In part A of Figure 1, the A section is ready for input (addresses 50-025 of the program loop in Table 3). At part B the B section outputs data, causing bit 7 of Control Register A to indicate that new data is available. CA2 goes high to inform the output device to send no more new information to avoid overrun error. Part C is where the CPU polls the 4PIO to see if it has new data to input. Part D is where the data is input. This clears bit 7 of register A and sets bit 7 of register B. The active high level of Control Register B bit 7 indicates to the CPU that new data can be output. CB2 will go low when new data is sent. This informs section A that new data is present. CB2 sets bit 7 of Control Register A by causing a transition on CA1. The cycle then repeats.

This information should help 4PIO users do extensive troubleshooting. But if you're still having problems with your 4PIO board, please contact your local Altair Computer Center or MITS.

TABLE 1

LOCATION	OP CODE	MENMONIC
000	333	IN, 40
001	040	
002	303	JMP
003	000	
004	000	

TABLE 2 FOR WAIT CIRCUITRY CHECK

IC	PIN	LABEL	A	B	C
N	6		P	H	L
G	9		P	L	H
G	10	SINP	P	L	H
O	1		P	H	L
O	7		H	H	H
O	3		L	L	L
C	14		P	H	H
C	13	PRDY	P	H	H
C	15		P	H	H
I	5		L	L	L
I	6	PWAIT	P	H	H
O	2		P	L	L
H	3	POC	H	H	H

P = PULSES, H=TTL HIGH (>2V.), L=TTL LOW (<.8V)

continued

GLITCHES
Troubleshooting the 88-4PIO

continued

TABLE 3

LOCATION	OP CODE	MNEMONIC	MEANING
000	076	MVI A	
	000		
	323	OUT,40	DONE TO TALK TO DDR OF A
	040		
	323	OUT,42	DONE TO TALK TO DDR OF B
	042		
	323	OUT,41	SET A UP AS ALL INPUTS
	041		
	076	MVI A	
010	377		
	323	OUT,43	SET B UP AS ALL OUPUTS
	043		
	076	MVI A	
	044		
	323	OUT, 42	TALK TO A'S DATA REGISTER
	042		
020	323	OUT, 40	TALK TO B'S DATA REGISTER
	040		
	006	MVI B	SET UP REGISTER B AS COUNTER
	000		
	333	IN,41	CLEAR BIT 7 OF A
	041		
026	333	IN, 43	CLEAR BIT 7 OF B
	043		
030	004	INR B	
	312	JZ	IF ALL COMBINATIONS TRIED STOP
	200		
	000		
	170	MOV A,B	
	323	OUT,43	OUTPUT BYTE TO B
	043		
	333	IN,40	CHECK TO SEE IF A RECOGNIZES
040	040		
	346	ANI	THAT DATA IS READY TO BE
	200		INPUTTED
	312	JZ	
	046		
	000		
	333	IN,41	INPUT THE BYTE FROM A
	041		
050	117	MOV C,A	SAVE BYTE TO BE OUTPUTTED
	220	SUB B	COMPARE IT TO BYTE OUTPUTTED
	302	JNZ	IF TWO DO NOT MATCH JUMP
	000		AND STORE BYTES
	010		
055	333	IN,42	SEE IF B RECOGNIZES THAT DATA
	042		WAS READ BY A
	346	ANI	
	200		
	312	JZ	
	064		
	000		
064	303	JMP	LOOP AROUND AND TRY
065	026		ANOTHER BYTE
	000		
000200	303	JMP	LOOP WHEN DONE
000201	200		
000202	000		
001000	171	MOV A,C	
001001	062	STA	ERROR ROUTINE
001002	100		STORE INPUTTED BYTE IN 100
001003	000		
001004	170	MOV A,B	
001005	062	STA	STORE OUTPUTTED BYTE IN 101
001006	101		
001007	000		
001010	303	JMP	LOOP TO SHOW THAT ERROR
001011	010		HAS OCCURRED
001012	010		

TABLE 4
CHECKING DATA LINES AND CONTROL SIGNALS

LABEL	IC	PIN	A	B	C	D	E
R/W	J	21	L	L	L	L	H
CS0	J	22	H	H	H	H	H
CS2	J	23	L	L	L	L	L
CS1	J	24	H	H	H	H	H
E	J	25	H	H	H	H	H
D7	J	26	L	L	L	H	*
D6	J	27	L	L	L	H	*
D5	J	28	L	L	L	H	*
D4	J	29	L	L	L	H	*
D3	J	30	L	L	L	H	*
D2	J	31	L	L	L	H	*
D1	J	32	L	L	L	H	*
D0	J	33	L	L	L	H	*
RST	J	34	H	H	H	H	H
RS1	J	35	L	H	L	L	L
RS0	J	36	H	H	L	L	L
	A	1	L	L	L	L	H
	B	1	L	L	L	L	H
	A	15	H	H	H	H	L
	C	1	H	H	H	H	L

* = VALUE MAY BE EITHER HIGH OR LOW

TABLE 5
ECHO WIRING FOR DB-25 CONNECTOR

PIN	LABEL	CONNECTED TO	PIN	LABEL
2	CA1		13	CB2
3	CA2		12	CB1
4	PA0		20	PB0
5	PA1		21	PB1
10	PB6		18	PA6
11	PB7		19	PA7
14	PA2		22	PB2
15	PA3		23	PB3
16	PA4		24	PB4
17	PA5		25	PB5

PIN NUMBERS REFER TO PINS ON DB-25 CONNECTOR

TABLE 6
6820 ECHO WIRING

LABEL	PIN	CONNECTED TO	LABEL	PIN
CA1	40		CB2	19
CA2	39		CB1	18
PA0	2		PB0	10
PA1	3		PB1	11
PB6	16		PA6	8
PB7	17		PA7	9
PA2	4		PB2	12
PA3	5		PB3	13
PA4	6		PB4	14
PA5	7		PB5	15

PIN NUMBERS REFER TO PINS ON 6820

Bits and Pieces

R/W Means Read It and Weep

By popular demand we're reprinting the following letter to the Editor (CN, August, 1975). It should provide some comic relief for those of you who are struggling through your first programs.

Wendell S. Rice
Chief Engineer
Data Documents Systems Corp.
Merriam, KS
Software Package No. 69
Altair SUPER EXTENDED BASIC - $1495.

When purchased with an Altair, 42K memory and either a duplex I/0 board and 4K of write-only memory, you have our deepest sympathy.

INSTRUCTION STATEMENTS:

CCS	Chinese Character Set
BH	Branch and Hang
BSO	Branch on Sleepy Operator
DO	Divide and Overflow
RPB	Reverse Parity and Branch
ARZ	Add and Reset to Zero
WWLR	Write Wrong Length Record
SRSD	Seek Record and Scar Disc
RC	Read Chaos
TDB	Transfer and Drop Bits
EROS	Erase Read Only Storage
UER	Update and Erase Record
CM	Circulate Memory
MWM	Move and Wrap Memory
DIA	Develop Ineffective Address
LMB	Lose Message and Branch
SC	Scramble Channels
LC	Loop Continuous
BIM	Branch on Index Missing
CD	Create Data
WOS	Write only Storage
BLI	Branch and Loop Indefinite
HCF	Halt and Catch Fire
BBI	Branch on Burned-out Indicator
BPO	Branch and Power-off
II	Inquire and Ignore
AI	Add Improper
ARZ	Subtract and Reset to Zero
RI	Read Invalid
WNR	Write Noise Record
ED	Eject Disc
EIOC	Execute Invalid Op Code
RNR	Read Noise Record
DSP	Destroy Storage Project
MDB	Move and Drop Bits
MLR	Move and Lose Record
MC	Move Continuous
RT	Reduce Thru-put
IOR	Illogical "OR"
IAND	Illogical "AND"
UCB	Uncouple CPU's and Branch
EO	Execute Operator
RBG	Random Bug Generator
RBG	Random Bug Generator (Special Feature)
IIB	Ignore Inquiry and Branch

CASH GRANTS OFFERED FOR SURVEYS OF PROGRESS IN ROBOTICS

The United States Robotics Society is offering three grants of $100 each to students who survey practical activity in research and development on robots in specified areas of the world. The surveys must be performed for academic credit with formal approval by appropriate professors.

With the sudden rise in the use of personal, privately owned computer systems, private research and development in robotics and artificial intelligence has surged. More than seventy members of USRS™ alone report active work on robots. The Society is seeking an estimate of robotics activity worldwide, and these first grants are the beginning of a general search for the robots.

Grants will be made for surveys of: The United States West of the Mississippi, the United States East of the Mississippi and Canada. Future grants will be made for surveys of other areas.

The reports will be published as part of the basic robotics literature, establishing their authors and supervisors as important contacts in the field.

Proposals from applicants are due on or before 30 September 1977. Completed reports are due on or before 30 June 1978.

For details, write:

Survey Grants
United States Robotics Society
Box 26484
Albuquerque, NM 87125

ASDC, AUG Scheduled to Move

The MITS Altair Software Division Company (ASDC) and The Altair User's Group (AUG) will move to Pertec's Microsystems Division (MSD) in Woodland Hills, California at the end of September.

Marion B. Guerin, who was recently named Sales Manager, Applications Software for Pertec's Microsystems Division (MSD), will be in charge of the move. "Most of the MSD marketing activities will be based in California," he said. "Having the software operation in close proximity to the other MSD marketing functions will make it easier to coordinate and improve the total marketing program." Guerin said this will mean more documentation, more support for dealers and just better overall software.

Guerin said the move will not delay any orders for programs.

CLASSIFIED ADS

FOR SALE:

Altair 8800a
Unused 1975 version, 256 bytes. Factory built and tested. $550. Ralph Reinke, 30305 Bob-O-Link Ave., Wausau, Wis. 54401. (715) 842-0196.

Altair 8800a CPU
$600.

88-4MCS 4K static memory card
$200

88-16MCS 16K static memory card
$800

88-DCDD disk drive and controller
$1500

88-2SIO serial teletype interface
$150

88-ACR audio cassette interface
$150

New ASR-33 Teletype
$1000

All above units brand new, factory assembled and tested. For more information about the above seven products, contact: Michael Clark, R.D. No. 3, Nazareth, PA. 18064. (215) 759-6873.

ROBOTS

AN ESTIMATE OF THE STATE OF THE ART, AND AN INVITATION TO PERSONS OF ADVENTUROUS SPIRIT AND INQUIRING MIND

We believe that the key discoveries necessary to the art of robotics have already been made. **We believe** that behind various national borders, behind the doors of various scientific disciplines from biochemistry to microelectronics, all of the primary technical obstacles have been overcome, all feasibilities been proven, all methods become known.

We believe that what remains to be achieved is principally the refinement of systems applying existing technologies — **and that this work proceeds apace.** We believe the world is about to encounter (where? when?) machines that truly simulate the intellectual and physical behavior of human beings: robots.

Robots are on our doorstep. Robots are almost within our reach. And we within theirs.

Robots are as frightening as they are alluring, as threatening as they are promising. Yet whatever reservation anyone may feel, there is now no turning back, no possibility of their denial or prohibition. The development of artificial intelligence proceeds not only in the laboratories of governments and industries, but also among the thousands of individual amateurs and hobbyists, free citizens exercising their freedom with experiments in the fascinating field of personal computing. **We believe** that since they are possible, robots are inevitable — "for good or ill."

The United States Robotics Society is established "for good" — for the good of mankind — not in opposition, for opposition is idle, and not in advocacy, for advocacy is unnecessary. **We invite** the support and active participation of all persons who can face the Age of the Robot with the appropriate curiosity and spirit of adventure.

Intelligent machines for production and service — tireless, able to understand commands and carry them out sensibly without feeling a need to make policy for themselves — may become the long-heralded boon to humanity, lifting ancient burdens of toil and suffering. But if they were to be developed "in the dark" — if they were to be sprung upon us full-blown, without our preparation — the reaction might be disastrous. The survival of our own society may depend quite soon (how soon?) on our ability to deal even with "friendly" robots. If we ignore them, if we are incompetent in their fields, we are surely not serving our own interests.

Intelligent weapons now appear practicable within the next decade or two — systems, for example, that can differentiate between friend and foe automatically, through their own sensors and judgement. If such weapons are developed anywhere in the world, they will be extraordinarily dangerous to any society which has not learned how to deal with them.

Robotics has charm not only for trained technicians and professionals but also for millions of persons without the skills and resources to participate directly in the work. Communication about robotics, like robots themselves, is inevitable, — through publicity, rumor, espionage, and now through The United States Robotics Society. This organization will assume the important task of identifying discoveries, gathering supporting data from the hidden recesses where they rest, collating, publishing, becoming a center of information for all parties seeking knowledge of current and historical activity in robotics. We urge you to be one of us — for just $12/year.

Benefits to USRS Members Growing Year by Year

- ● Certificate of Registration as USRS Member.
- ● USRS Newsletter, USRS bulletins, other correspondence from the Society as occasion demands.
- ● Aid in contacting other USRS Members in home regions, toward establishing USRS events.
- ● Opportunity (Qualified) to officiate as USRS Representative at Regional and National robotics shows and exercises.
- ● Service (Optional) as USRS Contributing Correspondent.
- ● Participation in the determination of procedures for investigating, reporting, archiving, and disseminating information relevant to robotics . . . and
- ● Privileged access to the Library of Robotics to be established by USRS.
- ● Discounts as may from time to time be arranged by USRS on behalf of members — with publishers, manufacturers/distributors of robotics related materials. (Note: this benefit alone can be expected to repay the moderate USRS Membership costs many times over.)

United States Robotics Society

A Non-Profit Organization
Glenn R. Norris, President
Box 26484 Albuquerque, New Mexico 87125

Application for Charter Membership

USRS

United States Robotics Society
Box 26484 Albuquerque, NM 87125

Enclosed is my check for $12 for enrollment and first-year dues.

NAME ______________________________

ADDRESS FOR USRS COMMUNICATIONS ______________________ CITY ____________ STATE __________ ZIP ________ PHONE ________

The following information is requested (OPTIONAL) to help ensure your full participation in the benefits of the Society.

My interest in robotics derives from ( ) intellectual curiosity ( ) academic training ( ) professional/business

Please tell us more: ______________________________

( ) I am interested in joining with others in local USRS activity.
I might serve as ( ) Correspondent ( ) Official at USRS functions.

When the chips are down . . .

and troubleshooting is in order, always refer to **Computer Notes.** In addition to articles that deal with troubleshooting procedures, **CN** carries information on the latest hardware, software and applications. **CN** is bound in a standard format that can be kept easily in a three-ring binder as a ready reference for the computer club or the individual user.

To insure that you never miss an issue, simply fill out the coupon below and send it along with the subscription fee to:

computer notes

mits a subsidiary of **Pertec Computer Corporation**
2450 Alamo S.E.
Albuquerque, New Mexico 87106

Please send me a 1 year subscription to **Computer Notes.**
$5.00 per year in U.S. $20.00 per year overseas.

NAME:__________

ADDRESS:__________

CITY:__________ STATE:__________ ZIP__________

COMPANY/ORGANIZATION__________

☐ Check Enclosed	MC or BAC/Visa #__________
☐ Master Charge	Exp Date __________
☐ BankAmericard/Visa	Signature__________

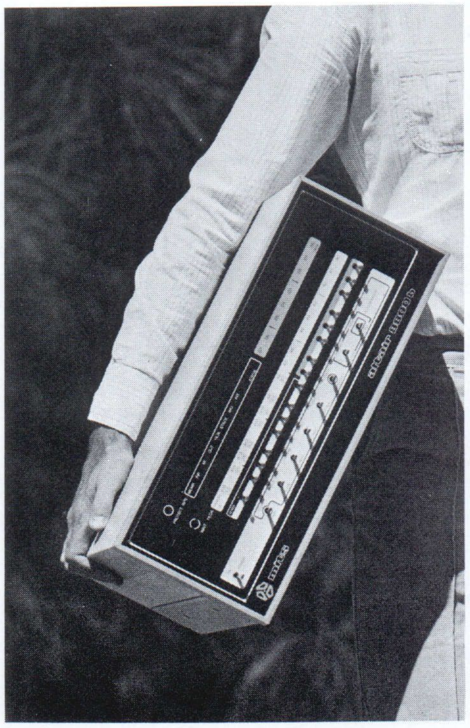
Computing has come a long way in the past three years. Computer systems, which previously were only attainable on a time-sharing or rental basis, have now become accessible to the general public. Innovative technology and proficient design are just some of the factors that spawned the growth of personal computers. The Altair™ 8800 microcomputer from MITS, Inc., was the initial result of these developments and the pacesetter of this new industry.

Altair Systems may be adapted for many diverse uses. We can assist and support your hardware/software needs in any aspect of computer operation. Altair micro computers have been utilized in research, educational programs, process and dedicated control as well as many original, user-generated applications.

Altair computer mainframes start as low as $395. Memory and interface capabilities are expandable through the use of Altair plug-in modules. Some of our recent additions permit process control, inexpensive mass storage and analog to digital conversion.

Even if you have never had any programming experience, Altair BASIC language can easily be learned and implemented. Within a short period of time, you will be solving complex problems without difficulty.

Altair microcomputer systems are readily available from any one of our nationwide dealers or through factory mail-order. Further information may be obtained by consulting your local Altair Computer Center or contacting us directly.



mits

The Personal Approach to Low cost Computing

MITS, Inc. 2450 Alamo, S.E. Albuquerque, N.M. 87106